Utility/helper thread API

Masamichi Takagi, Balazs Gerofi (RIKEN)
Rolf Riesen (Intel)
Kevin Pedretti (SNL)

r18

Revision history

Revision	Date	Description
r01	2016/07/26	RIKEN internal version
r02	2016/07/29	Merged Rolf’s email summary
r03	2016/08/09	Merged Aug 3rd discussion
r04	2016/10/03	Added draft API
r05	2016/10/19	Merged Oct 8 discussion
r06	2016/10/30	Merged Oct 30 discussion with ANL
r07	2016/11/14	Merged Nov 14 discussion with Michel, Kevin, Rolf, Balazs, Masamichi
r08	2016/12/07	Turned Motivation slide into Introduction and expanded; some edits for clarity on slides 6 - 7, 11 – 12, 16 – 17, 20, and 30. Also added some questions and notes as side-panel comments.
r08b	2016/12/07	Meeting: Masamichi, Balazs, and Rolf. Clarified attributes on slides 8 and 9. Expanded the description of some macros; needs more work. NUMA node -> domain. Some more clean up of the macro slides.

Revision history (continued)

Revision	Date	Description
r09	2016/12/12	Meeting: Rolf, Kevin, Balazs, and Masamichi. The function where a thread indicates itself is a utility thread (uti_pthread_indicate()) is moved to the "Future Work" part. The macros to specify a reference point in location (base_* macros) are moved to the "Considered but Dropped Ideas" part. Added more people to the acknowledgement slide. Added a con in slide 9. Upstreaming a new clone() is elaborated and moved to the "Future Work" part. Clarification in slide 5, 8, 10, 11, 13, 24, 30 and 32.
r10	2016/12/21	Meeting: Takayuki Okamoto (Fujitsu), Kouichi Hirai (Fujitsu), Atsushi Hori (RIKEN) Balazs and Masamichi. Add descriptions answering Fujitsu's question (slide 10, 18). [Proposal] Eliminate redundant Boolean arguments for location attribute macros (slides 20-23). Update design for McKernel (slides 30-32). Add discussion on providing ways to let runtime developer know the OS decision on location (slide 44). Add description on behavior when pthread_setaffinity_np() is used with location hints (slide 25). Add Fujitsu people to the acknowledgement slide (slide 4).
r11	2017/1/31	Add a way to switch libuti.so for Linux and McKernel (slide 39)
r12	2017/2/3	Meeting: Bob, Rolf, Dave, Tom, Yuaka, Balazs, Masamichi. Add executive summary (slide 4). Replace UTI_ATTR_CPU_SET with UTI_CPU_SET (slide 18, 30). Add implementation notes on switching library binaries between two OSes (slide 31-33).

Revision history (continued)

Revision	Date	Description
r13	2017/2/13	Merged Kevin's comment: Add configuration of Kitten (Slide 15,16). Correct typo (Slide 50).
r14	2017/5/1	Merged discussion on Apr 11: UTI_CPU_SET is only for Linux and Linux CPU ids are used.
r15	2017/6/16	Merged discussion on May 18: Repurpose UTI_CPU_SET to specify CPUs for utility thread. Add hints to prefer LWK core.
r16	2017/6/30	Merged two ideas of (1) reporting back to the caller if the hints are honored or not and (2) placing an utility thread on a CPU to which a fabric-related IRQ is routed, both of which are lead by John Attinella
r17	2017/7/12	Removed const in front of uti_attr in uti_pthread_creat()
r18	2019/2/22	Add environmental variable to specify CPUs for each process, including both utility CPUs and compute CPUs, to make it possible to spawn utility threads on compute CPUs, to describe locality and to provide load-balancing (draft)

Executive Summary

Background and purpose

- Various libraries and runtimes spawn utility/helper threads, leading to resource oversubscription which can hurt HPC application performance.
- The main problem is that the kernel cannot know whether a new thread is a utility thread or a computational thread.
- By providing an interface to let runtime systems give hints to the kernel when utility threads are being created the system can optimize their placement on the available resources.

Collaborators (OS developers)

- Intel, RIKEN, Sandia National Lab., Fujitsu

Target app/runtime/libraries

- Contacted, showed interest
  - MPICH, Open MPI, Intel OpenMP
- Potential users
  - IB drivers, High Performance ParalleX (HPX)

Introduction

- Various libraries and runtimes spawn asynchronous utility/helper threads
  - Example: asynchronous progress thread of MPICH, DAPL, monitor thread of Intel OpenMP, IB connection thread of Open MPI
- Especially for lightweight kernels with cooperative, non-preemptive schedulers these utility threads pose a big problem
- Thread oversubscription hurts performance
  - On Linux it can lead to extra context switches, it may imply frequent thread migrations and interference with application threads
  - On LWKs this issue becomes even worse, because:
    - LWKs usually run one thread per HW thread
    - Usually provide no timesharing
    - Co-operative scheduling may help, but needs explicit yielding from userland

Introduction (cont.)

- The main problem is that the kernel cannot know whether a new thread is a utility thread or a computational thread
  - Different utility threads have different requirements; e.g.
  - May need to run in same NUMA domain, poll frequently, can share a core with other utility threads, etc.
- If a utility thread can be identified and its requirements are known, the kernel can place them on an appropriate CPU
  - The special threads could be multiplexed over a set of dedicated physical resources
  - In multi-kernel OSes, helper threads could be placed on Linux cores

Desired solution

Part of the solution are two separate concerns/components:

1. Applications/runtimes/libraries should be able to indicate that a thread is not a computation thread
   - The API should be expressive enough to describe the thread’s particular attributes:
     - Some of these utility threads may need an entire logical CPU to themselves; maybe because they employ an MWAIT instruction or are polling so frequently that they consume the entire resource
     - Others are sleeping in poll()/select()/futex() calls most of the time
   - The API should be standardized among LWKs and Linux
   - Preferably with minimal changes to existing code
2. The ability to denote a set of resources (e.g., identify CPU cores or kernels) where utility threads should be scheduled
   - Some resources are reserved for system services in multi-kernel OS. When and how the reservation is performed is OS dependent, e.g. done at boot time by OS loader or done at job-launch time by node resource manager.

API alternatives

Timing

1. Parent marks utility thread during creation
2. Parent indicates that the child is a utility thread after creating it
3. Child itself indicates that it is a utility thread when it starts running

Abstraction level

A. clone() level

B. pthread_create() level

→ Support 1-B

Discussion on timing alternatives

Method	Pros	Cons
1. Parent indicates when spawning/creating the utility thread	• Eliminate additional system calls for migration • Eliminate interference with compute-threads, e.g. disturbing CPU binding of compute-threads	• Need to modify both clone() and pthread_create() functions when supporting both of the abstraction levels
2. Parent indicates after spawning utility thread	• Less intrusive for application code, i.e. one additional function call • One function could support both of the abstraction levels	• Caller should pass tid to specify the child thread but there is no clean way because pthread_t is an opaque type • Costly, hard-to-implement migration across partition boundary is needed in partitioned multi-kernel OS (e.g. McKernel)
3. Child indicates after it has been spawned	• Same as above	• Costly, hard-to-implement migration across partition boundary is needed in partitioned multi-kernel OS (e.g. McKernel)

Location Attributes

- Hints on resource allocation and scheduling
  - OS would place a new utility thread on one of the Linux CPUs or the CPUs dedicated for system services even when not passing any hints.
- Any hints can be ignored by the implementation
- Attributes are implemented as an opaque type and users manipulate it by a set of macros

	Name	Type	Description
1	cpu_set	Set of integers	Specify the set of CPUs used for utility threads. The format is comma-separated list of Linux CPU ids and id-ranges, as in cpulist_parse(). Example: 0-2, 4, 6 ==> {0, 1, 2, 4, 6}
2	numa_domain_set	Set of integers	Request to place the thread on one of the NUMA domains in this set. The set contains NUMA domain numbers. It is assumed that NUMA domains are numbered by the same method the underlying OS uses.
3	same_numa_domain	bool	Request to place the thread in the same NUMA domain as the caller is in.
4	different_numa_domain	bool	Request to place the thread in a different NUMA domain than the caller is in.
5	same_{L1,L2,L3}	bool	Request to place the thread on the CPU sharing L1/L2/L3 cache with the caller.
6	different_{L1,L2,L3}	bool	Request to place the thread on a CPU not sharing L1/L2/L3 cache with the caller.
7	prefer_{COMP,SYS}	bool	Request to place the thread on a compute/system service CPU.
8	fabric_intr_affinity	bool	Request to place the thread on a CPU to which a fabric (interconnect) related IRQ is routed to

Behavior Attributes

	Name	Type	Description
7	needs_exclusive_CPU	bool	This utility thread needs to be the sole runnable thread on a hardware thread. Perhaps because it calls a thread-blocking MWAIT.
8	CPU_intensive	bool	This utility thread uses a lot of CPU cycles and is sensitive to disruptions. Perhaps because it is polling frequently.
9	high_priority	bool	Use high priority scheduling for this utility thread. It can run alongside others, but needs to be given higher priority.
10	non_cooperative	bool	This utility thread will not yield the CPU very frequently. For other tasks to run, the scheduler needs to preempt this utility thread.

Draft API for apps/runtime/libraries

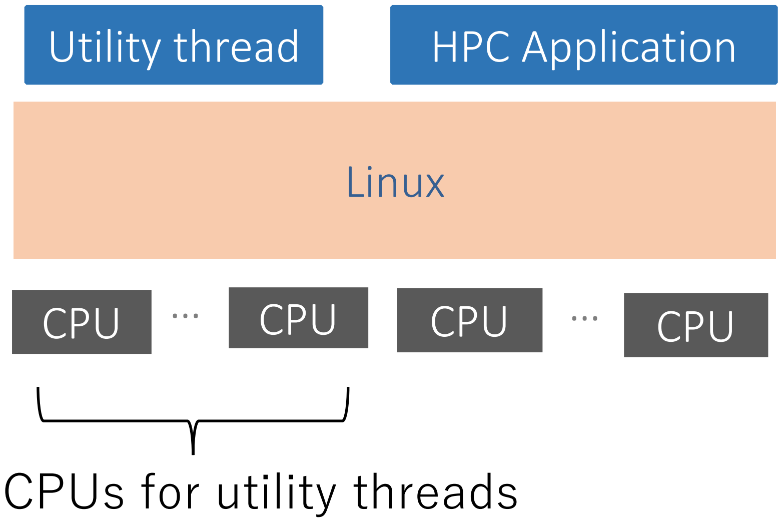
Goals

1. Enable applications/runtimes/libraries to indicate that a thread is not a computation thread
2. Enable applications/runtimes/libraries to denote the location (e.g. which core or kernel) where utility threads should be placed or the behavior that lets the kernel know what it can and should do with it

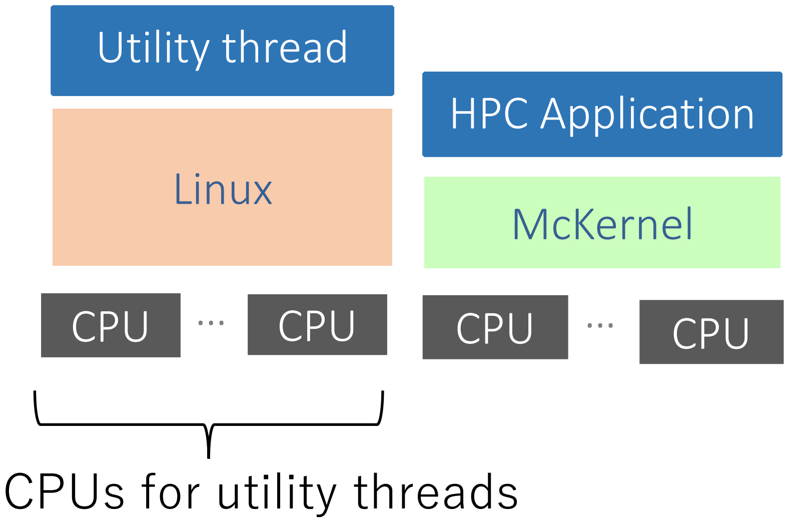
Usage

Discussion needed

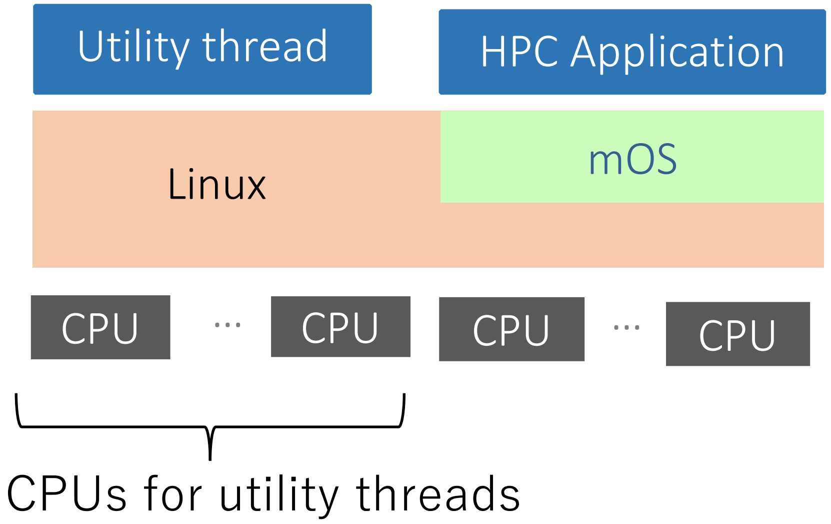
Regular Linux



McKernel

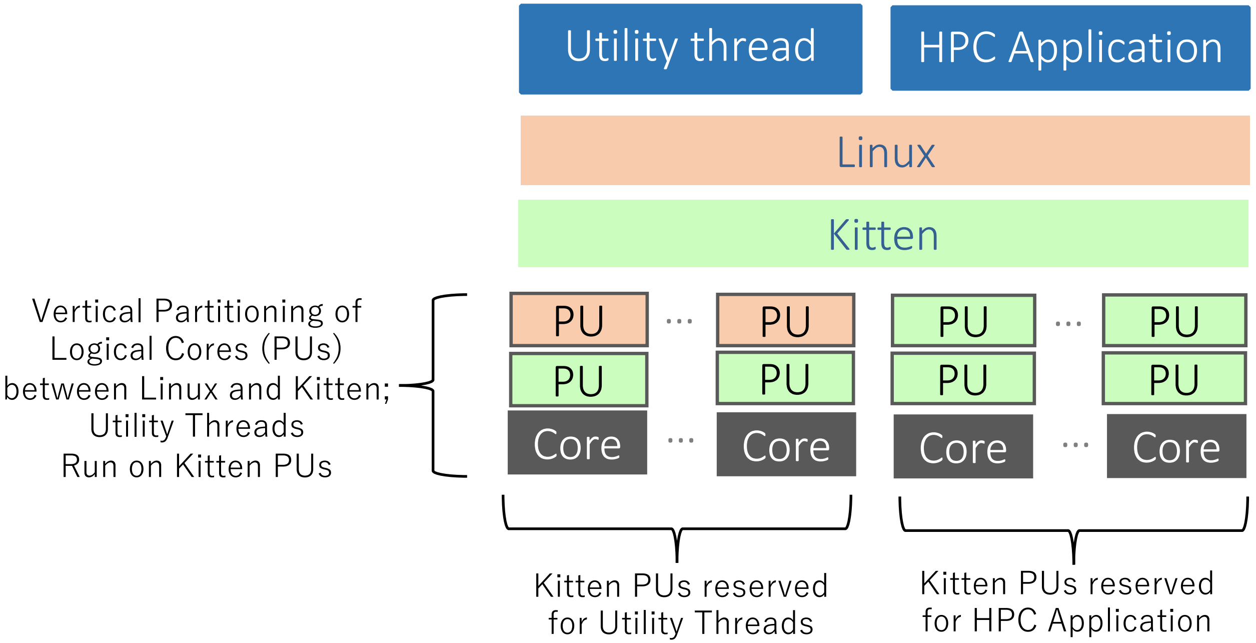


mOS



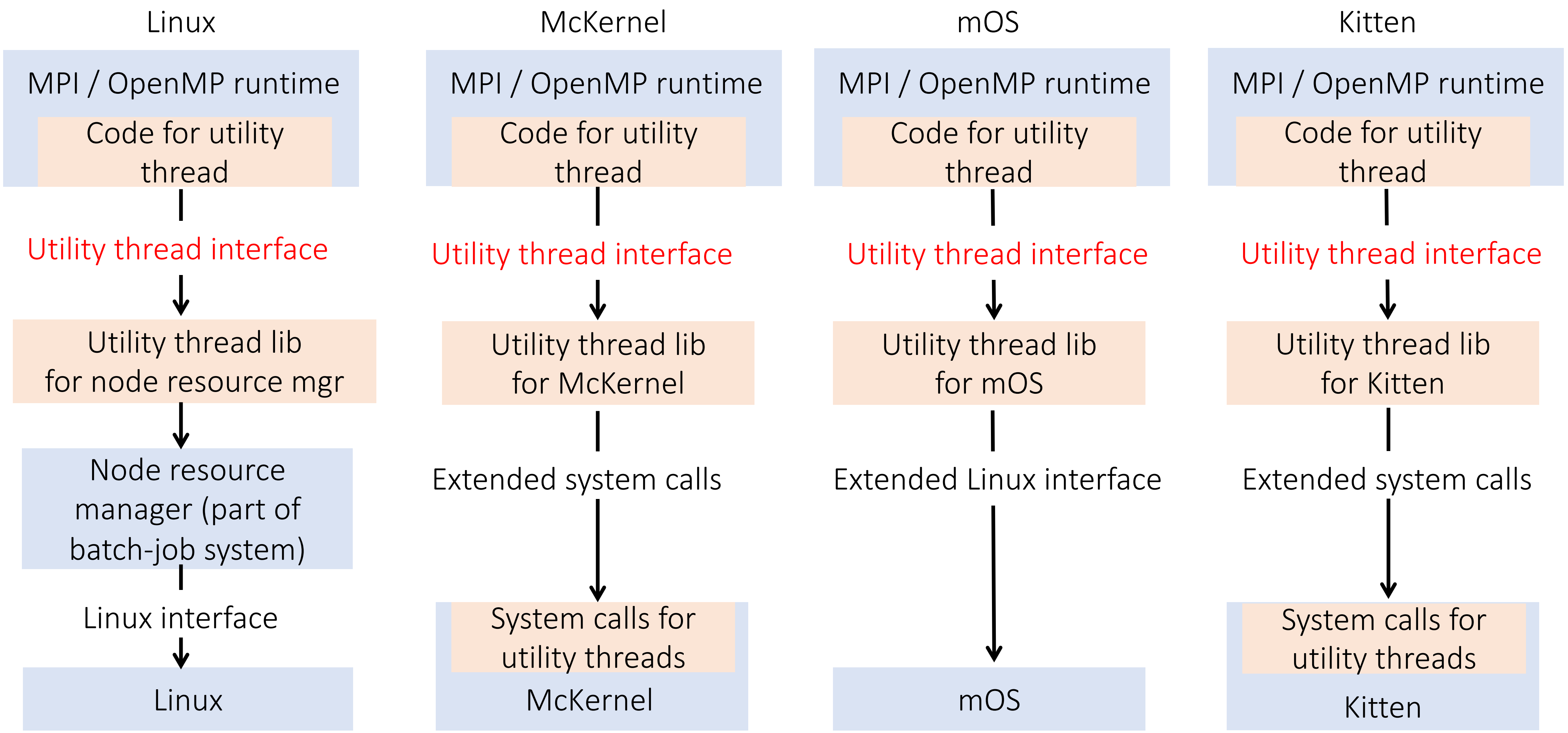
1. (Linux only) Specify system service CPUs
2. (Partitioned OS only) Specify CPUs for each process
3. Prepare an attribute describing the location and behavior of the utility thread
4. Create the pthread-compatible thread by passing the attribute
5. (Optional) Check if the attributes are honored or not

Kitten (Hobbes Multi-Kernel)


Utility thread
HPC Application
Linux
Kitten
PU ··· PU PU ··· PU
PU PU PU PU
Core ··· Core Core ··· Core
Vertical Partitioning of Logical Cores (PUs) between Linux and Kitten; Utility Threads Run on Kitten PUs
Kitten PUs reserved for Utility Threads
Kitten PUs reserved for HPC Application

Software Components and Interface

The functions of Utility Thread Interface (UTI) are provided to apps/runtimes/libraries in the form of user-level library



uti_attr_init Function

Synopsis

int uti_attr_init(uti_attr_t *uti_attr)

Description

Initialize the object of utility thread attributes pointed to by uti_attr. All of the attributes are set to invalid.

Return value

0	Success
EINVAL	uti_attr is invalid

uti_attr_destroy Function

Synopsis

int uti_attr_destroy(uti_attr_t *uti_attr)

Description

Destroy the object of utility thread attributes pointed to by uti_attr. Destroying an object of utility thread attributes has no effect on threads that were created using the object.

Return value

0	Success
EINVAL	uti_attr is invalid

Discussion needed

UTI_NODE_SYS_CPUS Environmental Variable

Variable Name

UTI_NODE_SYS_CPUS

Description

This variable denotes the system service CPUs available to utility threads for a node. The format is comma-separated list of Linux CPU ids and id-ranges, as in cpulist_parse(), e.g. 1-3,5 means {1, 2, 3, 5}.
OS with partitioning (e.g. mOS, McKernel) sets this variable automatically.

Example:

- CPU #0-3 belong to numa-node #0, CPU #4-7 to numa-node #1
- CPU #1-3 and #5-7 are used for McKernel
- UTI_CPU_SET should be "0, 4"

Discussion needed

UTI_PROC_UTIL_CPUS Environmental Variable

Variable Name

UTI_PROC_UTIL_CPUS

Description

This variable denotes the CPUs available to utility threads for a process. It can include the compute CPUs as well as the system service CPUs. The format is the same as UTI_NODE_SYS_CPUS. A CPU for a utility thread is chosen from the set in a load-balanced way. Linux sets this variable automatically to the CPU affinity. McKernel sets this variable to the CPU affinity plus IKC routing target CPUs.

Example:

- CPU #0-3 belong to numa-node #0, CPU #4-7 to numa-node #1
- CPU #1-3 and #5-7 are used for McKernel
- CPU affinity of process #0 is set to #1-3 and process #1 to #5-7
- UTI_PROC_UTIL_CPUS of process #0 should be "0-3", #1 should be "4-7"

UTI_ATTR_NUMA_SET Macro

Synopsis

int UTI_ATTR_NUMA_SET(uti_attr_t *uti_attr, unsigned long *nodemask, unsigned long maxnode)

Description

Set numa_set attribute of uti_attr to the set specified by nodemask and maxnode and make the attribute valid. nodemask points to a bit vector whose length is maxnode. A utility thread created with this attribute is meant to run within the indicated NUMA domain.

Return value

0	Success
EINVAL	uti_attr is invalid

UTI_ATTR_SAME_NUMA_DOMAIN Macro

Synopsis

int UTI_ATTR_SAME_NUMA_DOMAIN(uti_attr_t *uti_attr)

Description

Set same_numa_domain attribute of uti_attr to true and make the attribute valid.
This macro is used to indicate that the utility thread should run in the same NUMA domain as the caller is in.

Return value

0	Success
EINVAL	uti_attr is invalid

UTI_ATTR_DIFFERENT_NUMA_DOMAIN Macro

Synopsis

int UTI_ATTR_DIFFERENT_NUMA_DOMAIN(uti_attr_t *uti_attr)

Description

Set different_numa_domain attribute of uti_attr to true and make the attribute valid. This macro is used to indicate that the utility thread should not run in the same NUMA as the caller is in.

Return value

0	Success
EINVAL	uti_attr is invalid

UTI_ATTR_SAME_{L1,L2,L3} Macro

Synopsis

```
int UTI_ATTR_SAME_{L1,L2,L3}(uti_attr_t *uti_attr)
```

Description

Set same_{l1,l2,l3} attribute of uti_attr to true and make the attribute valid. This macro is used to indicate that the utility thread should run in the same vicinity as the caller is in.

Return value

0	Success
EINVAL	uti_attr is invalid

UTI_ATTR_DIFFERENT_{L1,L2,L3} Macro

Synopsis

int UTI_ATTR_DIFFERENT_{L1,L2,L3}(uti_attr_t *uti_attr)

Description

Set different_{l1,l2,l3} attribute of uti_attr to true and make the attribute valid. This macro is used to indicate that the utility thread should not run in the same vicinity as the caller is in.

Return value

0	Success
EINVAL	uti_attr is invalid

UTI_ATTR_PREFER_{LWK,FWK} Macro

Synopsis

int UTI_ATTR_FABRIC_INTR_AFFINITY(uti_attr_t *uti_attr)

Description

Set prefer_{lwk,fwk} attribute of uti_attr to true and make the attribute valid. This macro is used to indicate that the utility thread should be placed on a LWK/FWK CPU.

Return value

0	Success
EINVAL	uti_attr is invalid

UTI_ATTR_FABRIC_INTR_AFFINITY Macro

Synopsis

int UTI_ATTR_FABRIC_INTR_AFFINITY(uti_attr_t *uti_attr)

Description

Set fabric_intr_affinity attribute of uti_attr to true and make the attribute valid. This macro is used to indicate that the utility thread should be placed on a CPU to which fabric-(interconnect)-related IRQ is routed.

Return value

0	Success
EINVAL	uti_attr is invalid

Utility Thread Behavior Attributes Macros

Synopsis

```
int UTI_ATTR_EXCLUSIVE_CPU(uti_attr_t *uti_attr)
int UTI_ATTR_CPU_INTENSIVE(uti_attr_t *uti_attr)
int UTI_ATTR_HIGH_PRIORITY(uti_attr_t *uti_attr)
int UTI_ATTR_NON_COOPERATIVE(uti_attr_t *uti_attr)
```

Description

Set the corresponding attribute of uti_attr to true and make the attribute valid. This macro is used to describe the behavior of the utility thread. See page 10 for details.

Return value

0	Success
EINVAL	uti_attr is invalid

uti_pthread_create Function

Synopsis

```
int uti_pthread_create(
        pthread_t *thread, const pthread_attr_t * attr, void *(*start_routine) (void *), void * arg,
        uti_attr_t *uti_attr)
```

Description

Create a pthread-compatible non-computation thread and denote the utility thread attributes specified by uti_attr. When pthread_setaffinity_np() is used with the location attributes, OS tries to find a CPU which is a member of both the CPU set specified by the attributes and the set specified by pthread_setaffinity_np().

Return value

0 on success; an error number on error.

Errors

EAGAIN Insufficient resources to create another thread, or a system-imposed limit on the number of threads was encountered.

EINVAL Invalid settings in attr. This includes the case when pthread_setaffinity_np() is used with the location attributes and OS failed to find a CPU which is a member of both the CPU set specified by the attributes and the set specified by pthread_setaffinity_np().

EPERM No permission to set the scheduling policy and parameters specified in attr.

Macros for checking whether or not attributes are honored

Synopsis

```
int UTI_RESULT(uti_attr_t *uti_attr)
int UTI_RESULT_NUMA_SET(uti_attr_t *uti_attr)
int UTI_RESULT_{SAME,DIFFERENT}_NUMA_SET(uti_attr_t *uti_attr)
int UTI_RESULT_{SAME,DIFFERENT}_{L1,L2,L3}(uti_attr_t *uti_attr)
int UTI_RESULT_PREFER_{LWK,FWK}(uti_attr_t *uti_attr)
int UTI_RESULT_FABRIC_INTR_AFFINITY(uti_attr_t *uti_attr)
int UTI_RESULT_EXCLUSIVE_CPU(uti_attr_t *uti_attr)
int UTI_RESULT_CPU_INTENSIVE(uti_attr_t *uti_attr)
int UTI_RESULT_HIGH_PRIORITY(uti_attr_t *uti_attr)
int UTI_RESULT_NON_COOPERATIVE(uti_attr_t *uti_attr)
```

Description

The first macro returns whether or not all of the attributes passed are honored or not. Each of the following macros returns whether or not the corresponding attribute is honored or not. UTI_RESULT_NUMA_SET returns "honored" if the CPU location is within the set.

Return value

1	The attribute is honored
0	The attribute is not honored
EINVAL	uti_attr is invalid

Usage Example

Async. Progress Thread of MPICH: Module Structure

MPICH provides its own thread creation function and uses it when creating an asynchronous progress thread

Call graph

```
MPI_Init()
   MPIR_Init_async_thread()
      MPID_Thread_create()
         MPIDU_Thread_create()
            MPL_thread_create()
```

Async. Progress Thread of MPICH: Code

```
void MPL_thread_create(MPL_thread_func_t func, void *data, MPL_thread_id_t * idp, int *errp){
   ...
#if MPL_THREAD_PACKAGE_NAME == MPL_THREAD_PACKAGE_UTI
                uti_attr_t uti_attr;
                err = uti_attr_init(&uti_attr);
                if(err) { goto uti_exit; }

                /* Suggest that it's beneficial to put the thread on the same NUMA-domain as the caller */
                err = UTI_ATTR_SAME_NUMA_DOMAIN(&uti_attr);
                if(err) { goto uti_exit; }

                /* Suggest that the thread repeatedly monitors a device */
                err = UTI_ATTR_CPU_INTENSIVE(&uti_attr);
                if(err) {   goto uti_exit; }

                err = uti_pthread_create(idp, &attr, MPLI_thread_start, thread_info, &uti_attr);
                if(err) { goto uti_exit; }

                err = uti_attr_destroy(&uti_attr);
                if(err) { goto uti_exit; }
         uti_exit:;
#else
        err = pthread_create(idp, &attr, MPLI_thread_start, thread_info);
#endif
   ...
}
```

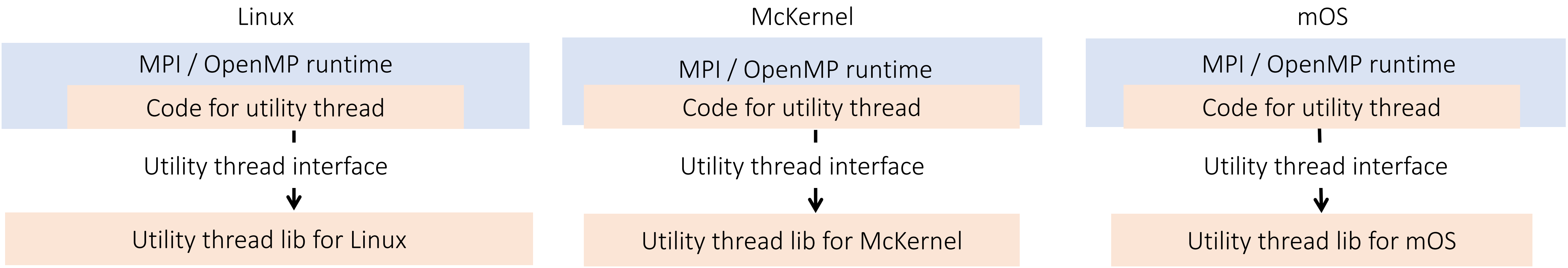
Implementation Notes

Reserving CPUs for utility thread

- There are two ways for a system to reserve CPUs for utility thread
- UTI library for each system should instruct in its document app/library/runtime developer how to do it.

System	Way for OS to reserve CPUs	Instruction to user
• McKernel • mOS • Linux which reserves CPUs using cgroup	OS reserves them at its boot time and set them to UTI_CPU_SET	Basically a user doesn't need to change the value of UTI_CPU_SET. He/she can change when a fine-tuning is needed.
Regular Linux	OS doesn't reserve them and doesn't set a value to UIT_CPU_SET	A user need to specify the CPUs by using UTI_CPU_SET. He/she need to set the same value to I_MPI_PIN_PROCESSOR_EXCLUDE_LIST when using Intel MPI library.

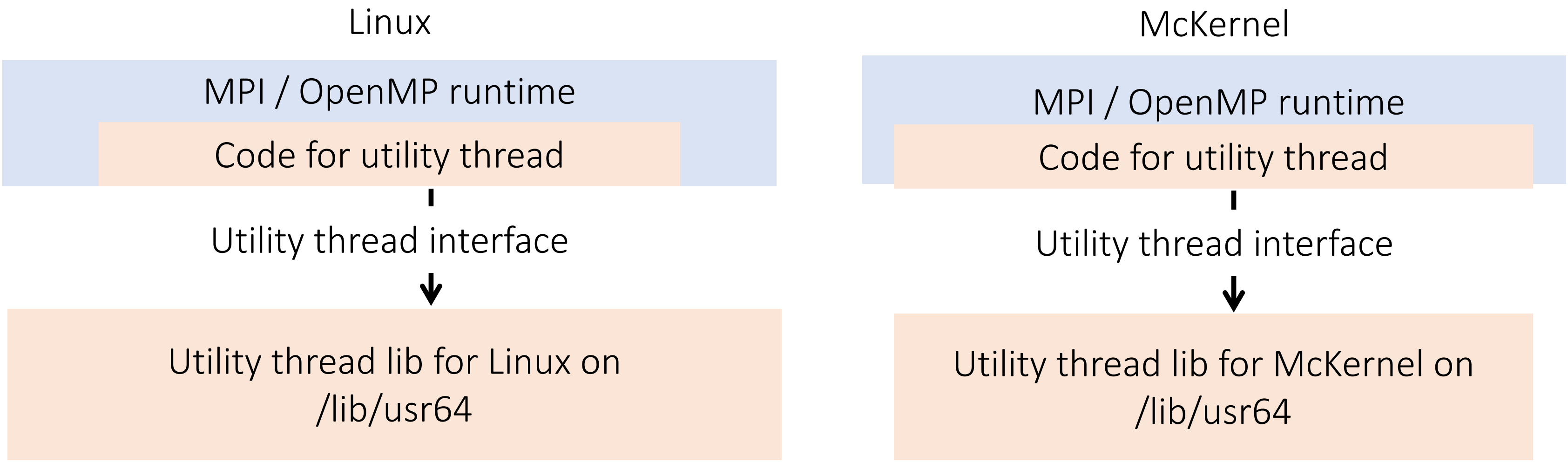
Switching OS-specific Utility Thread Libraries (1/3)



We want the same app executable which work on two OSes (e.g. McKernel and Linux) and need to link to the OS specific library in a user-transparent way.

Switching OS-specific Utility Thread Libraries (2/3)

Example of McKernel and Linux



Linux

MPI / OpenMP runtime

Code for utility thread

Utility thread interface

Utility thread lib for Linux on /lib/usr64

McKernel

MPI / OpenMP runtime

Code for utility thread

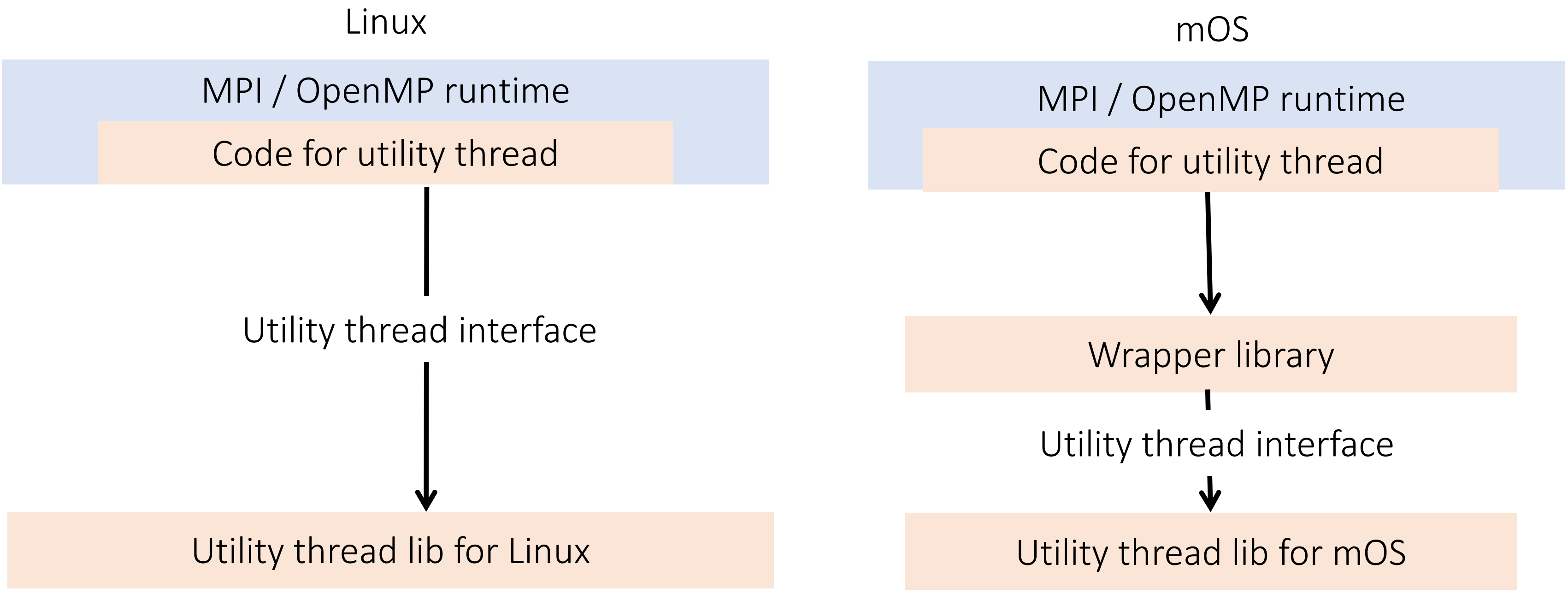
Utility thread interface

Utility thread lib for McKernel on /lib/usr64

Loader mcexec bind-mounts McKernel specific UTI library implementation onto the standard library path.

Switching OS-specific Utility Thread Libraries (3/3)

Example of mOS and Linux



Linux

MPI / OpenMP runtime

Code for utility thread

Utility thread interface

Utility thread lib for Linux

mOS

MPI / OpenMP runtime

Code for utility thread

Wrapper library

Utility thread interface

Utility thread lib for mOS

Job launcher yod relinks wrapper library against mOS specific UTI library implementation.

Action Items

- Provide a reference implementation for regular Linux which don't use any partitioning techniques such as cgroup

Implementation and Evaluation

McKernel Design - Thread Management (1/3)

Approach

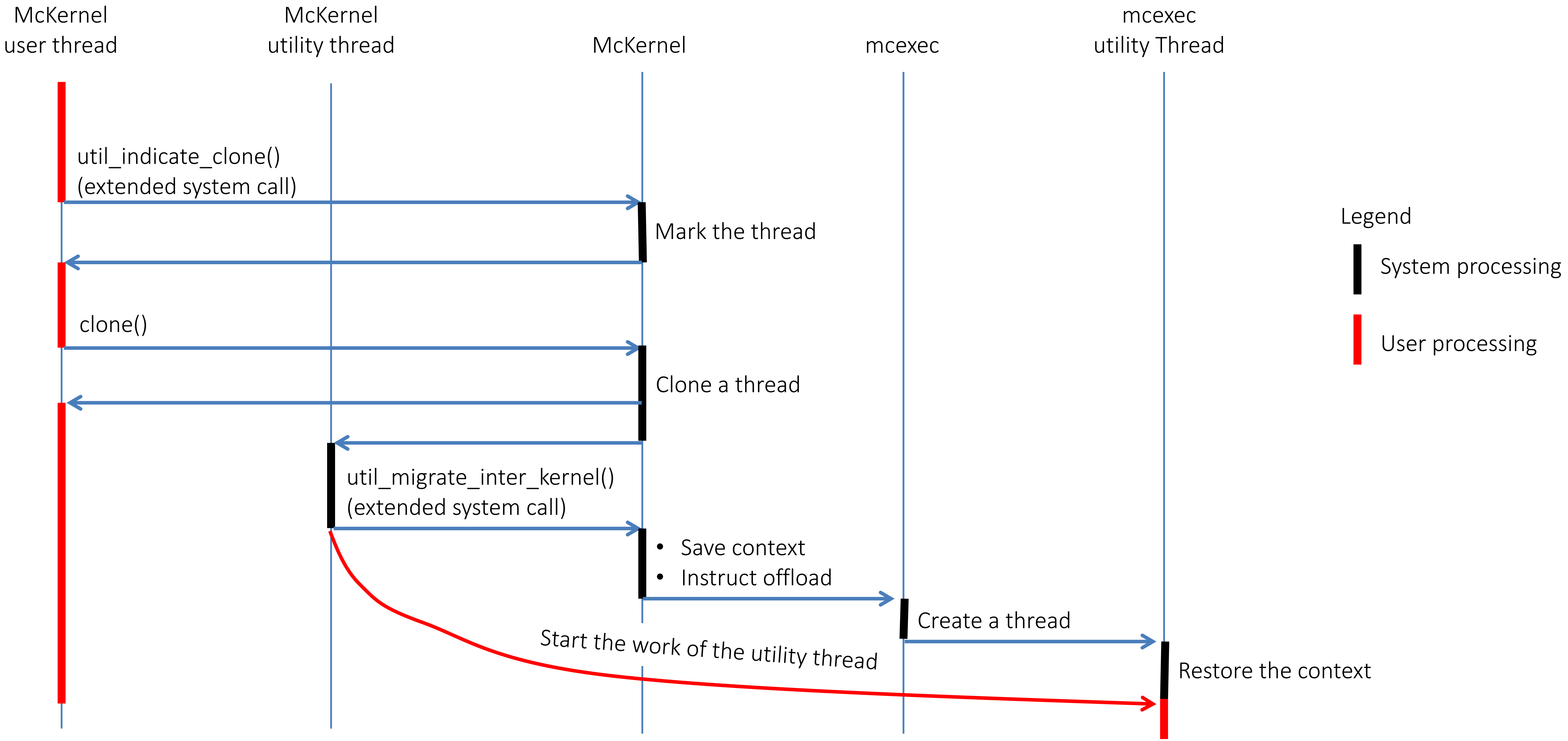
- Run the utility thread in Linux in the glibc context of the McKernel process
  - Offload from Linux to McKernel (call it reverse-offload) system calls which manipulate the context of the McKernel process (e.g. mmap)
  - Prevent TLS of the Linux thread from getting corrupted in the signal handler which is shared with the other mcexec threads
- Reverse-offload futex() to synchronize with threads in McKernel

Steps

1. The user process indicates that the next clone() from the McKernel side creates a utility thread
2. McKernel spawns a pthread on McKernel
3. The McKernel thread spawns the actual utility pthread on Linux
4. McKernel copies the context of the McKernel thread to the Linux thread
5. Reverse-offload brk(), m[un]map() and futex()
6. Relay a signal sent from McKernel threads targeted at the utility thread to the Linux side
7. Save/restore TLS (part of the McKernel thread context) when the mcexec signal handler is called while in the context of the Linux thread

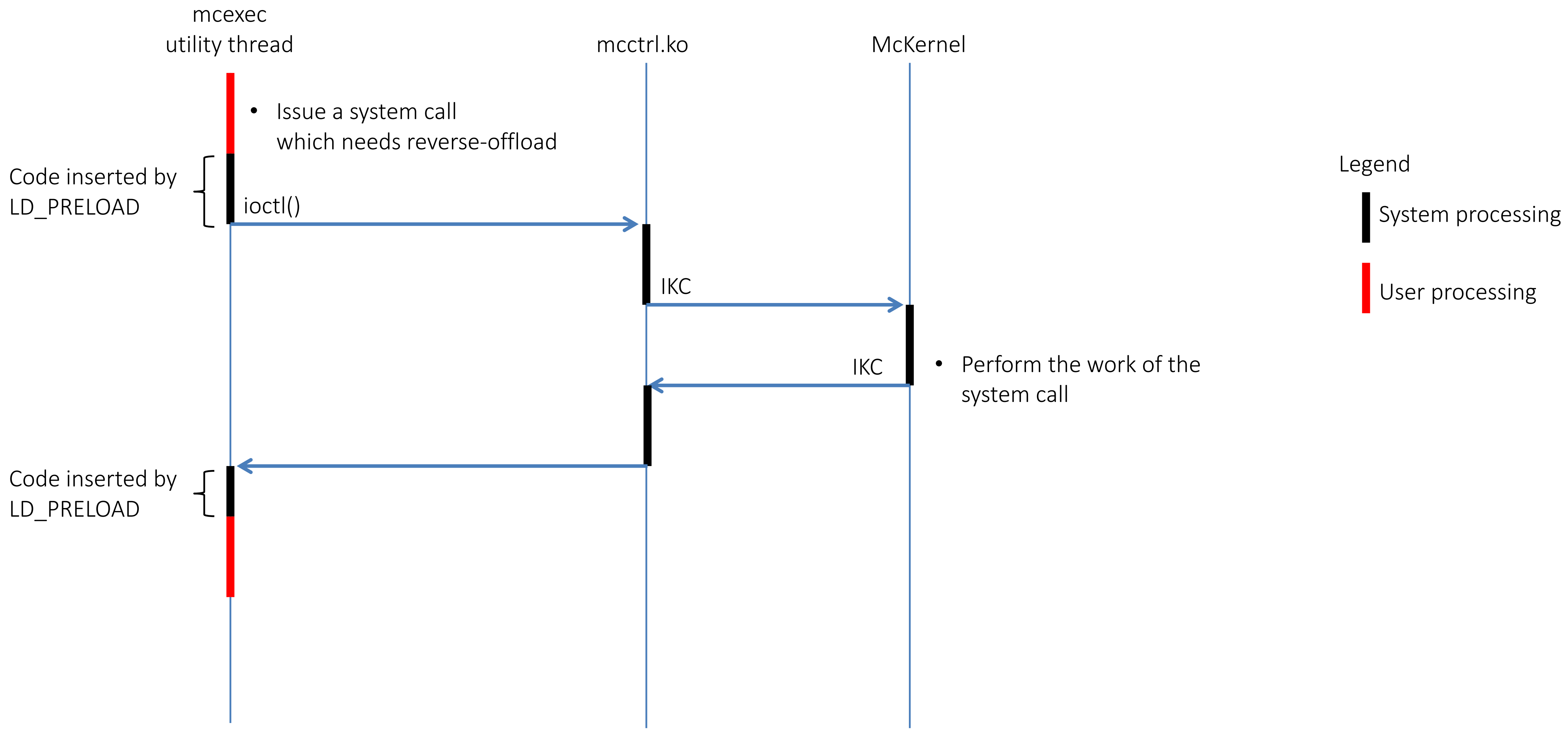
McKernel Design - Thread Management (2/3)

Flow: Spawn a pthread from McKernel to Linux



McKernel Design - Thread Management (3/3)

Flow: Reverse-offload system calls



mcexec utility thread

mcctrl.ko

McKernel

- Issue a system call which needs reverse-offload

Code inserted by LD_PRELOAD

ioctl()

IKC

IKC

- Perform the work of the system call

Code inserted by LD_PRELOAD

Legend

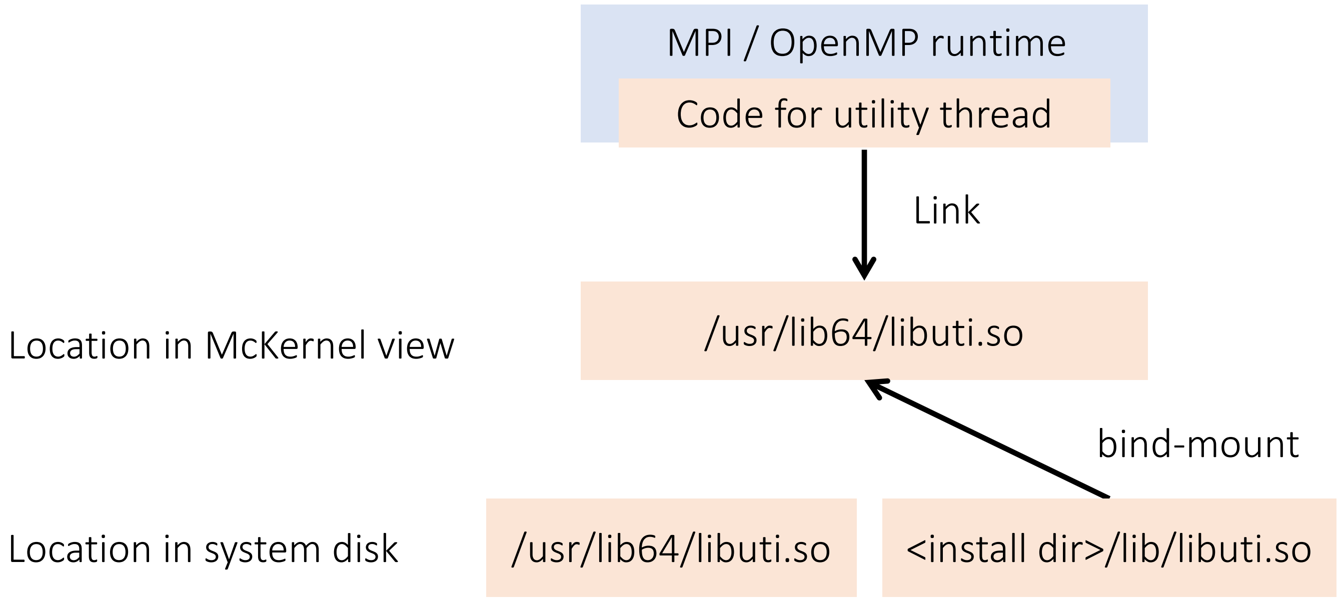
■ System processing

🟥 User processing

McKernel Design - Switching OS-Specific UT library

Steps

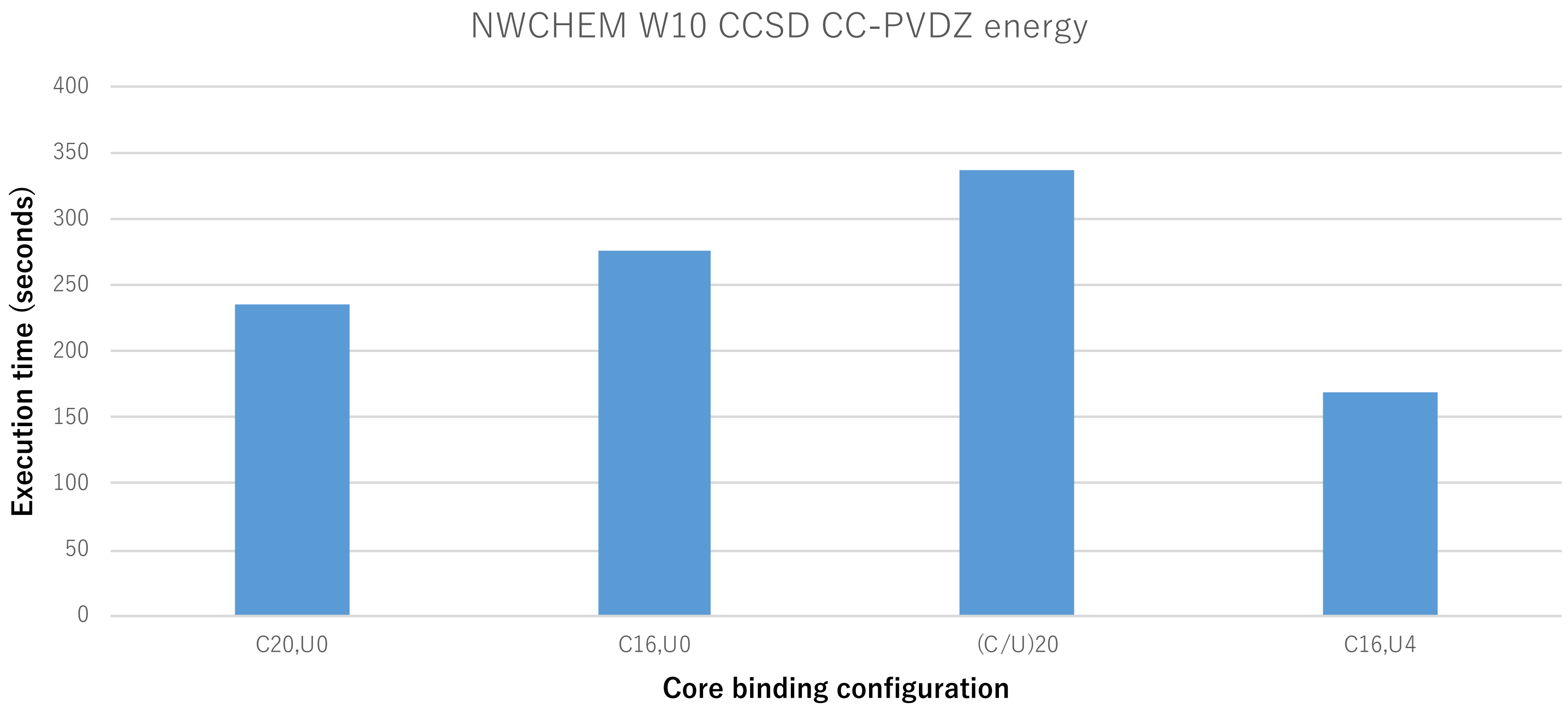
- Call the library libuti.so
- mcexec performs bind-mount <install dir>/lib/libuti.so on /usr/lib64/libuti.so



mOS Design

- (To be filled in by John)

Preliminary Evaluation on MPI Progress Thread Awareness



- NWCHEM
  - 32 nodes, 2-sockets 10 cores / socket Xeon and Mellanox Connect-IB
  - MVAPICH2-2.1 (with modification on progress thread)
- Four core binding configurations:
  - C20,U0 means 20 ranks without progress thread
  - (C/U)20 means 20 ranks with 20 progress threads, each progress thread runs on the same core as its parent MPI rank
  - C16,U4 means 16 ranks with 16 progress threads, 4 cores are dedicated to progress threads, each progress thread runs on the same NUMA domain as its parent MPI rank
- TODO: measurements should be also done with hyperthreading on and using AMG miniApp

Discussion Detail

Discussion on Implementation Alternatives of Type 1-B (1/3)

1. Indicate via a clone-level creation function

Pros

- Can create the thread directly on Linux
- Allow different implementations of creation function with the standardized interface (e.g. clone_util() in the figure)

Cons

- It is difficult to build the pthread-compatible glibc structures

Solutions

- Create a new thread using the special clone-level function
- Let the thread create a pthread-compatible one
- Return the thread ID of this new thread to the caller

clone() to target core and then pthread_create() (mOS)

```
pthreadize(..., entry, ...) {
   pthread_create(..., entry, ..);
      /* Prepare pthread compatible glibc structures */
}

uti_pthread_create(..., uti_attr_t *uti_attr) {
   ... /* pthread_create() code */
   clone_util(..., entry, ..., uti_attr, ...);
     /* Special system call with standardized interface
        which creates a thread on Linux core and
        jumps to pthreadize() */
   ... /* pthread_create() code */
}
```

Discussion on Implementation Alternatives of Type 1-B (2/3)

2. Indicate with a notify-function preceding the create-function

Pros

- Can create the thread directly on Linux
- It is easy to create pthread-compatible glibc structures by reusing the code of pthread_create()

Cons

- Might need to modify the thread creation function so that it understands the notification

Notifying the next clone() is a special one (mOS)

```
uti_pthread_create(..., uti_attr_t *attr) {
   ... /* pthread_create() code */
   util_indicate_clone();
      /* Special system call which notifies that next clone()
         should create a utility thread */
   clone();
      /* Modified system call which puts the thread on
         Linux core when notified */
   ... /* pthread_create() code */
}
```

Bind core and pthread_create() (Linux)

```
uti_pthread_create(..., uti_attr_t *attr) {
   pthread_attr_setaffinity_np(attr, ...); /* Unmodified */
   pthread_create(..., attr, ...); /* Unmodified */
}
```

Discussion on Implementation Alternatives of Type 1-B (3/3)

3. Create a thread on LWK and then migrate

Pros

- It is easy to create pthread-compatible glibc structures by reusing the code of pthread_create()

Cons

- Cannot create the thread directly on Linux, so incurs additional overhead and disturbs CPU binding of compute-threads

Solutions

- Treat the sleeping thread differently when trying to find vacant cores for compute-threads

clone() in sleep state and then migrate (McKernel)

```
uti_pthread_create(..., uti_attr_t *attr) {
   ... /* pthread_create() code */
   clone_sleep(...);
     /* Special system call which creates a sleeping thread on
        McKernel core */
   migrate_inter_kernel(..., attr, ...);
     /* Special system call which migrates it to Linux core */
   ... /* pthread_create() code */
}
```

pthread_create() and then indicate (mOS)

```
wrapper(..., entry) {
   util_indicate(...);
      /* Special library function which migrates it to Linux core */
    (*entry)(...)
}
uti_pthread_create(..., uti_attr_t *attr) {
   pthread_create(&pthread, ..., wrapper, ...); /* Unmodified */
}
```

pthread_create() and migrate it with cgroup (Linux)

```
uti_pthread_create(..., uti_attr_t *attr) {
   pthread_create(...); /* Unmodified */
   migrate_cgroup(..., tid, ...);
     /* Function provided by resource manager which migrates
        it to system service cgroup */
}
```

Discussion on Attributes

	Name	Description
1	cpu_set	(1)-(2) are useful when you already resolved processor topology. (1)-(2) and (3)-(7) are usually used exclusively. It specifies a set of CPUs instead of a CPU to follow the Linux way (i.e. sched_set_affinity()) though a user often would specify only one CPU. cpu_set cannot be used when CPUs are numbered in a different way than Linux in the kernel of the caller. It can't be used when the Linux cores are not visible to the kernel.
2	numa_domain_set	-
3	same_numa_domain	(4)-(7) are useful when you don't want to resolve processor topology.
4	different_numa_domain	-
5	same_{L1,L2,L3}	-
6	different_{L1,L2,L3}	-

Future Work

Support of Post-creation Self-indication

- Adding 3-B (i.e. the caller indicates it is a utility thread, see page 7)

Discussion

1. Supporting both of 1-B and 3-B makes the API more complicated and makes the usage less uniform because they are quite different
2. Implementing 3-B is difficult
3. No use-case of 3-B is found so far

→ Won't pursue it unless the runtime community really starts pushing for it

Upstream a New clone()-level Function to Linux Community

Possible Plan

- Propose extensions to clone() to the Linux community
  - With showing the benefit
- Propose extensions to pthread_create() to the pthread community
  - Extensions to pthread_attr
  - A new pthread_create() which can understand the attribute and use the new clone()
  - With showing an example implementation of the pthread_create()

Letting developers know the decision on location

- Runtime developers might want to know the decision done by OS
- We can add macros to obtain the decision (an example for CPU set is shown below). However, the utility thread can find it with system calls (e.g. getcpu())

UTI_ATTR_GET_CPU_SET Macro

Synopsis

```
int UTI_ATTR_GET_CPU_SET(uti_attr_t *uti_attr, size_t cpusetsize, cpu_set_t *cpuset)
```

Description

Get the cpu location chosen by OS to cpuset. cpusetsize denotes the size of cpuset.

Return value

0	Success
EINVAL	uti_attr is invalid
ENODATA	The attribute is not set

→ Won't pursue it unless the runtime community really starts pushing for it

Considered but Rejected Ideas

Setting a Reference Point for Location Attributes (1/4)

Purpose

- Allow users to set the reference point for {same,different}_* attributes so that they can indicate such as:
  - Place the thread in the same NUMA domain as CPU #18
  - Place the thread on a core sharing L2 with CPU#18

Attribute added/modified

Name	Type	Description
base_{cpu,numa,node}	Integer	Specify the base CPU/NUMA domain for{same,different}_*. The CPU/NUMA domain of the caller is used as the base when this attribute is not valid.
same_numa_domain	bool	Request to place the thread in the base NUMA domain.
different_numa_domain	bool	Request to place the thread in a the different NUMA domain than the base NUMA domain.
same_{L1,L2,L3}	bool	Request to place the thread on the CPU sharing L1/L2/L3 cache with the base CPU.
different_{L1,L2,L3}	bool	Request to place the thread on the CPU not sharing L1/L2/L3 cache with the base CPU.

Setting a Reference Point for Location Attributes (2/4)

Macros added

UTI_ATTR_BASE_{CPU,NUMA,Node} macro

Synopsis

```
int UTI_ATTR_BASE_CPU(uti_attr_t *uti_attr, int cpu)
int UTI_ATTR_BASE_NUMA(uti_attr_t *uti_attr, int cpu)
int UTI_ATTR_BASE_Node(uti_attr_t *uti_attr, int cpu)
```

Description

Set the base attribute of uti_attr and make the attribute valid. This is the base CPU, NUMA domain, or node that is used as a reference for the UTI_ATTR_SAME_x and UTI_ATTR_DIFFERENT_x macros.

Return value

0	Success
EINVAL	uti_attr is invalid

Setting a Reference Point for Location Attributes (3/4)

Macros changed: UTI_ATTR_{SAME,DIFFERENT}_*

- Showing the case for UTI_ATTR_SAME_NUMA_DOMAIN and the similar changes are applied to others

UTI_ATTR_SAME_NUMA_DOMAIN macro

Synopsis

```
int UTI_ATTR_SAME_NUMA_DOMAIN(uti_attr_t *uti_attr, bool same)
```

Description

Set same_numa_domain attribute of uti_attr to same and make the attribute valid.
This macro is used to indicate which NUMA domain the utility thread should run in. If a base domain or base CPU has been set, then that is used as the reference point. Otherwise the same NUMA domain that the caller is in, is chosen.

Return value

0	Success
EINVAL	uti_attr is invalid

Setting a Reference Point for Location Attributes (4/4)

Discussion

- Make the API complex and inconsistent
  - The intention of {same,different}_* attribute was to allow users to specify location in an abstracted way (complementing {cpu,numa_domain)_set which require exact locations) but base_* require user to specify exact locations
- It is difficult for users to specify exact locations
  - Because the resources given to Linux/LWK might change site-by-site, OS-by-OS, job-by-job
  - It's even impossible when the resources are invisible to the caller
- It is difficult for users to predict the behavior because the attributes might be ignored
- Make the implementation complex
  - It needs to calculate the union of the reference CPU-set and NUMA-domain-set and then narrow down the candidates using behavior when both are specified

→ Won't include this idea for now

UTI_ATTR_CPU_SET Macro

Synopsis

int UTI_ATTR_CPU_SET(uti_attr_t *uti_attr, size_t cpusetsize, cpu_set_t cpuset)

Description

Set cpu_set attribute of uti_attr to the set specified by cpusetsize and cpuset and make the attribute valid. cpusetsize denotes the size of cpuset.

When a new utility thread is created with this attribute, it indicates to the kernel that it should run on one of these CPUs.

Return value

0	Success
EINVAL	uti_attr is invalid

This macro won't work for the system where Linux CPUs are invisible to the caller (e.g. McKernel).

→ Replaced with UTI_CPU_SET environmental variable

Acknowledgements

Acknowledgements

We received valuable feedback from:

- Abdelhalim Amer (ANL, MPICH developer)
- John Attinella (Intel, mOS developer)
- Pavan Balaji (ANL, MPICH developer)
- Michael Blocksome (Intel, MPICH developer)
- George Bosilica (UTK, Open MPI developer)
- Kouichi Hirai (Fujitsu, OS developer)
- Krishna Kandalla (Cray, MPICH developer)
- Tom Musta (Intel, mOS developer)
- Lena Oden (ANL, MPICH developer)
- Takayuki Okamoto (Fujitsu, OS developer)
- Kenneth Raffenetti (ALN, MPICH developer)