CMPT 300 — Operating Systems I

Summer 1999

Segment 4:

Processes and Threads

Melissa O'Neill

User's view of Processes

- A Fundamental OS abstraction
- Vary in details from OS to OS:
  - Batch system — jobs
  - Time-shared systems — user programs or tasks
- Common idea — process = a program in execution
- A process includes:
  - program
  - current program state
    (i.e., data in memory, processor state, etc.)

User's view of Processes (contd.)

- Under Unix, processes:
  - are created with fork, exit with exit
  - have events signalled with kill
  - wait for children to finish with wait

  but operating systems vary in the details of how process creation etc. work…

- On any multiprogrammed system, multiple processes may be active at any one time (c.f., uniprogrammed system)
  - even if *really* the processor can only physically be running instructions from one process at any given time

- Processes have a degree of independence from each other
  - may only communicate through designated communications methods
  - one errant processes should not affect other unrelated processes

User's view of Processes (contd.)

- The environment you interact with is made up of processes

```
USER       PID  %CPU %MEM VSIZE RSIZE TT STAT  TIME COMMAND
melissa    217   8.8 16.2 33.7M 5.17M ?  SW    25hr WindowServer
root       230   6.2  0.8 1.61M  248K p1 SW    41hr top
root       228   2.1  5.1 22.5M 1.64M ?  SW  239:28 Terminal
root     15400   1.1  0.6 2.09M  200K p9 R   525:50 httpd
melissa    241   0.8  1.1 1.68M  352K p7 SW    0:29 csh
melissa  10674   0.3  2.0 5.66M  640K ?  SW  123:35 Preferences
melissa    223   0.2  5.3 6.18M 1.71M ?  SW    5:41 WM
root        25   0.0  0.9 6.73M  296K ?  S    50:52 nmserver
root        92   0.0  0.7 1.58M  232K ?  SW    2:59 syslogd
root        97   0.0  0.6 1.57M  208K ?  SW    0:37 portmap
root       100   0.0  0.5 1.67M  152K ?  SW    2:20 routed
root       104   0.0  0.7 1.59M  240K ?  SW    0:26 nibindd
root       105   0.0  1.1 1.63M  368K ?  SW    3:41 netinfod
root       109   0.0  1.0 1.69M  312K ?  SW    2:58 lookupd
root         3   0.0  0.4 3.26M  120K ?  SW    0:02 kern_loader
root       113   0.0  0.2 1.65M   80K ?  S     0:00 biod
root       115   0.0  0.2 1.65M   80K ?  S     0:00 biod
root       116   0.0  0.2 1.65M   80K ?  S     0:00 biod
root       126   0.0  0.6 1.61M  192K ?  SW    0:38 autonfsmount
root       132   0.0  0.2 1.68M   80K ?  S     0:00 bootpd
root       135   0.0  0.3 1.58M   96K ?  SW    0:00 rpc.bootparamd
root       141   0.0  0.7 1.58M  224K ?  SW    0:15 inetd
root         2   0.0  0.3  688K   88K co SW    3:26 mach_init
root       114   0.0  0.2 1.65M   80K ?  S     0:00 biod
root       163   0.0  0.3 2.12M   96K ?  SW    0:00 lpd
root       173   0.0  1.4 3.79M  448K ?  SWN   0:13 npd
root       177   0.0  1.7 1.96M  552K ?  SW    2:48 named
nobody     180   0.0  0.4 1.72M  120K ?  SW    5:31 ssockd
root       189   0.0  0.5 1.64M  168K ?  SW    2:04 aarpd
root       191   0.0  0.5 1.66M  160K ?  SW    1:06 atis
root         1   0.0  0.1  736K   40K ?  SW    0:04 init
root       202   0.0  0.4 1.77M  136K ?  SW    0:42 sshd1
root       209   0.0  0.4 1.65M  120K ?  SW    7:01 update
root       212   0.0  0.5 1.65M  152K ?  SW    3:19 cron
root       218   0.0  1.0 5.73M  344K ?  SW    0:02 loginwindow
melissa    219   0.0  1.4 3.81M  456K ?  SW    0:38 pbs
melissa    221   0.0  0.6 2.13M  200K ?  SW    0:01 appkitServer
root       145   0.0  0.8 1.88M  264K ?  S     0:20 sendmail
```

```
(contd.)
USER       PID  %CPU %MEM VSIZE RSIZE TT STAT   TIME COMMAND
root       150   0.0  0.3 2.13M  112K ?  SW     0:00 lpd
melissa    225   0.0  5.8 11.4M 1.84M ?  RWN 122:27 Mail
melissa    226   0.0  3.4 5.55M 1.08M ?  SW  212:09 BackSpace
root       160   0.0  0.3 1.91M   96K ?  SW     0:00 pbs
melissa    231   0.0  0.4 1.77M  128K p2 SW    76:04 tail
melissa    232   0.0  0.6 1.68M  184K p3 S      0:05 csh
melissa    235   0.0  0.4 1.68M  144K p4 S      0:06 csh
melissa    239   0.0  1.0 1.68M  312K p6 SW     0:14 csh
root       194   0.0  0.5 1.64M  160K ?  SW     0:56 snitch
melissa    243   0.0  0.5 1.68M  168K p8 S      0:16 csh
melissa    507   0.0  0.4 1.77M  128K p7 SW    74:27 tail
melissa    585   0.0  1.0 3.94M  312K ?  SW     0:34 nextspell
melissa   1675   0.0  6.1 28.9M 1.95M ?  SW     0:17 Webster
melissa  11225   0.0  1.6 11.8M  528K ?  SW     0:24 Librarian
melissa  11836   0.0  0.6 4.17M  184K ?  SWN    0:01 PDFCryptServer
melissa  11838   0.0  0.9 4.12M  296K ?  SWN    0:04 FlateFilterServe
melissa  11840   0.0  0.9 4.09M  296K ?  SWN    0:01 PDFFontFileServe
root     12011   0.0  0.5 3.67M  168K ?  SW     0:02 Faxxess
root     12012   0.0  0.7 1.60M  232K ?  SW     0:08 TrimProgram
root     12525   0.0  0.9 1.76M  288K ?  SW    10:59 aufs
root         0   0.0 14.9 17.7M 4.76M ?  R N  710hr kernel idle
root       222   0.0  3.7 6.02M 1.17M ?  S N    1:16 Workspace
root     10380   0.0  0.3 1.80M   88K ?  S      0:00 plug-gw
melissa  12554   0.0  1.5 1.69M  496K ?  SW     1:46 fetchmail
nobody   13952   0.0  0.7 2.09M  232K p9 S      0:03 httpd
nobody   13955   0.0  0.7 2.02M  232K p9 SW     0:03 httpd
melissa  17200   0.0  0.5 1.68M  160K p5 S      0:00 csh
melissa  29082   0.0  2.0 4.84M  640K ?  SW     0:38 Preview
melissa   2214   0.0  5.6 4.80M 1.80M ?  SW     3:02 WriteNow
melissa   3753   0.0  1.6 1.91M  528K ?  SW     1:47 aufs
melissa   3757   0.0  0.9 1.66M  280K p6 T      0:00 ncftp
root      4131   0.0  0.6 1.02M  184K Ca SW     0:00 pppd
root      4253   0.0  0.7 1.69M  240K p9 SW     0:01 telnetd
melissa   4254   0.0  1.0 1.68M  312K p9 S      0:02 csh
melissa   4316   0.0  5.9 5.38M 1.88M ?  SW     0:04 Edit
melissa   4319   0.0  3.3 2.70M 1.07M p6 S      0:01 kermit
melissa   4321   0.0  0.8 2.70M  272K p6 S      0:00 kermit
nobody    4337   0.0  0.8 1.64M  264K ?  SW     0:00 ssockd
```

- about 75 processes executing on my workstation
- times represent 35 days of uptime

Process Concept

- A process has two aspects:
  - resource ownership (memory, files, etc.)
  - despatching (processor use)
- Active processes must share the available resources

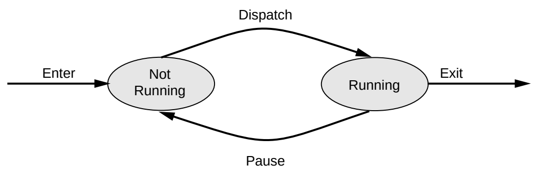
Process Implementation

- How does the OS implement the process abstraction?

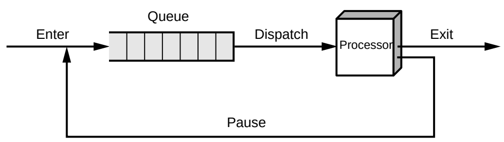
Process Implementation

- Maintain *process image* for each process
  i.e., storage containing:
  - program code
  - program data
  - processor stack
  - housekeeping information (PCB)
- Switch CPU between active processes (*process switch*)

A Two-state Process Model



(a) State transition diagram



(b) Queuing diagram

- But what exactly is in the queue here?

Process Control Block

Class Exercise: What information should be stored in a process control block?

Process Switching

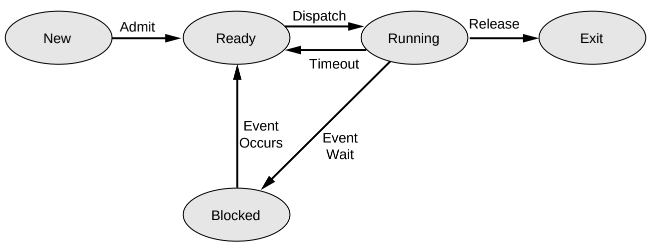
Class Question: When can/do we switch processes?

Process Switching

We could switch processes any time the OS has control, i.e.,

- interrupt occurs
  - clock
  - I/O interrupt
  - memory fault
- Trap occurs
  - trace
  - protection fault
- System call
  - I/O request
  - wait for child
  - *etc.*

A Five-state Process Model



- Admit (and release) are operations supported by the *long-term scheduler*
- Other operations are supported by the *short-term scheduler*

A Five-state Process Model

Possible states for a process…

- ***New***: Process is being created.
  - No resources allocated yet.
- ***Running***: Instructions are being executed.
- ***Ready***: Process is waiting to be assigned to a processor.
- ***Blocked***: Process is waiting for some event to occur.
- ***Exited***: Process has finished execution.

Class Question: Why do we need an "exited" state?

Process Scheduling Queues

The machine keeps track of which processes are in which states using queues.

- ***New queue*** — processes waiting to be created
- ***Ready queue*** — processes (residing in main memory), ready and waiting to execute.
- ***Event queues*** — processes waiting for a particular event (e.g., waiting for an I/O request to complete)

Tip: In C++, we would say something like:

```
queue< ProcessStartInfo * >   new_queue;
Process *                     current_process
queue< Process * >            ready_queue;
queue< Process * >            tape_drive_queue;
```


Schedulers

- ***Long-term scheduler*** — selects which processes should be brought into the ready queue.
  - Invoked very infrequently (seconds or minutes)
  - May be slow
  - Controls the degree of multiprogramming
- ***Short-term scheduler*** — selects which process should be executed next and allocates it to the processor.
  - Invoked very frequently (milliseconds)
  - Must be fast

Process Switch

When system switches to another process, the system must save the state of the old process and load the saved state for the new process.

- Process-switch time is overhead
- Time required depends on hardware support.

Cooperating Processes

Two possibilities:

- ***Independent processes*** cannot affect or be affected by the execution of another process.
- ***Cooperating processes*** can affect or be affected by the execution of another process.

Advantages of process cooperation:

- Information sharing
- Computation speed-up
- Modularity
- Convenience

Sharing Stateful Resources

Share any resource that has a readable and settable state:

- Memory
- Files

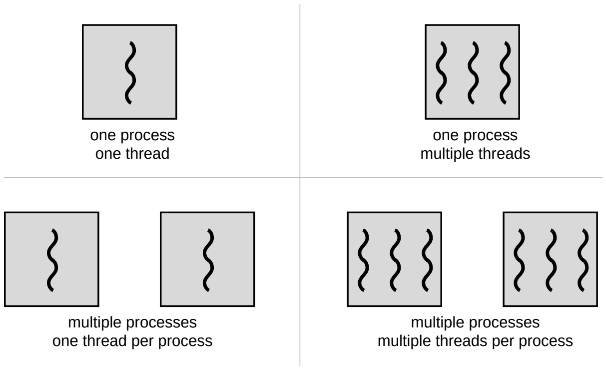
Threads

- Traditional processes
  — Virtual uniprocessor machine
- Multithreaded processes
  — Virtual multiprocessor machine

Thus, threads

- Share
  - Address space (i.e., memory)
  - Other resources (e.g., open files)
- Don't share
  - Processor registers
  - Processor stack area

Threading Possibilities



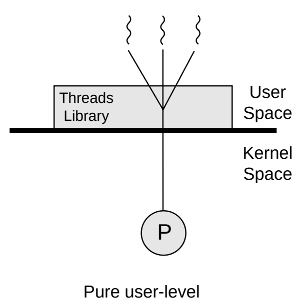
Uses of Threads

- Performing foreground and background work
- Supporting asynchronous processing
- Speeding execution
- Organizing programs

Class Exercise: Can an application implement threads without thread support being built into the OS?

If so, what *does* it need from the from the OS to support threads?

Model for User Threads



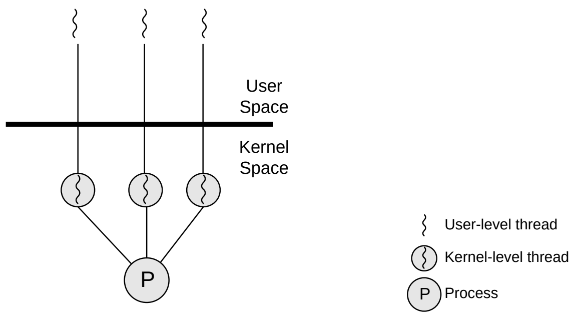
- No kernel overhead for thread library calls
- Scheduling policy in thread library can be quite different from of kernel — can be application specific

But,

- I/O issues
- Can't take (easily) take advantage of multiprocessing

Model for Kernel-level Threads

- So, maybe we should put the threads in the kernel?



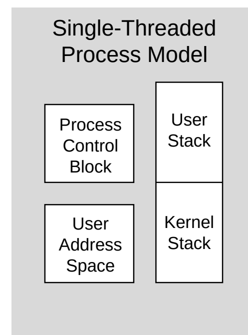
Pure kernel-level

But,

- Now we have kernel overheads
  - Kernel data structures
  - Mode switch to kernel

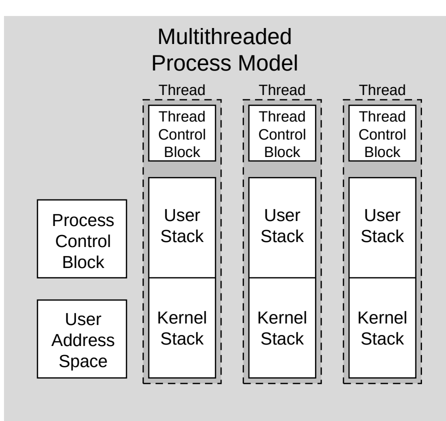
Changes to Process Control Block to Support Kernel-level Threads

- Before



Changes to Process Control Block to Support Kernel-level Threads

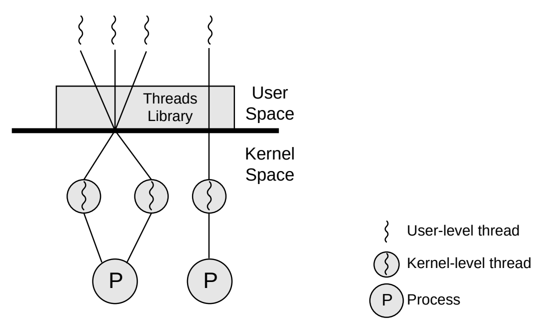
- After



- What is in a thread control block? Typically just registers.

Hybrid schemes for threads

- This approach is taken by Solaris, which calls its kernel-level threads *lightweight processes*



Combined

Scheduler Activations

A better way to deal with kernel/user thread package issues:

- Kernel talks to the thread library through *upcalls*
- An upcall is a call from the kernel to a user code
  - Kernel creates a new kernel thread (called a *scheduler activation*)
  - Calls routine in user thread library
- Upcalls happen:
  - When a thread blocks
  - When a thread becomes ready

Message-Based Interprocess Communication (IPC)

Messages are an alternative to communication though shared memory or shared files.

- Analogous to sending a message by mail, or a package by sea
- Provides a virtual communications medium
- Requires two basic operations:
  - send_message(*destination*, *message*)
  - receive_message(*sender*, *message*)

Class Exercise: The above definitions of send_message and receive_message are remarkably vague.
What details are missing?
What are the options?

Implementation Questions

- Is a “connection” set up between the two processes?
  - If so, is the link unidirectional or bidirectional?
- How do processes find the addresses of their friends?
- Can many processes send to the same destination?
- Can many processes receive at the same destination?
- Does the sender wait until the receiver receives the message?
- Does the receiver always know who sent the message?
- Can the receiver restrict who can talk to it?
- What is the capacity of the receiver’s mailbox?
- Is the recipient guaranteed to be on the same machine?
- Can messages be lost?
- Can messages vary in size or is the size fixed?
- Do messages contain typed data?

Example: Message passing in Mneme (ports.cc)

- *Basics*

  Mneme calls its message sources and destinations “ports” (the sea analogy).

  There are two classes:
  - LocalPort (where messages are send and received)
  - RemotePortAddress (names for places that messages can be sent to)

- *Is a “connection” set up between the two processes?*

  No. Mneme uses connectionless datagrams.

- *Can a process have more than one LocalPort?*

  Yes.

- *How do processes find the addresses of their friends?*

  Mneme ports are named using ASCII strings (filenames!).

Example (continued)

- *Can many processes send to the same destination?*

  Yes.

- *Can many processes receive at the same destination?*

  No.

- *Does the sender wait until the receiver receives the message?*

  No. (But if the destination mailbox is full, the process will block until the message can be placed in the mailbox).

- *Does the receiver always know who sent the message?*

  Usually. (It is possible to create anonymous LocalPorts, but this is rarely done.)

Example (continued)

- *Can the receiver restrict who can talk to it?*

  Only by receiving messages, checking who they are from, and throwing away ones that are from "undesirable" senders.

- *What is the capacity of the receiver's mailbox?*

  Approximately 32 KB of data.

- *Do messages arrive in order?*

  Messages from the same sender arrive in order. Messages from different senders may not arrive in the order they were sent.

- *Is the recipient guaranteed to be on the same machine?*

  Yes, currently.

Example (continued)

- *Can messages be lost?*

  Not under NEXTSTEP, Linux or Solaris 2.6 for local delivery.

- *Can messages vary in size or is the size fixed?*

  Message size can vary. Large messages (more than 10 KB) may cause problems and are not supported.

- *Do messages contain typed data?*

  No. The messaging primitives see messages as a simple byte sequence.

  (But the MessageBuffer class provides a mechanism for extracting typed data from a sequence of bytes).

- *What happens if the receiver dies?*

  Messages already delivered to the receiver's mailbox will be lost. Otherwise, a system_error exception will be thrown.

Class Exercise: Using the Mneme port classes as a foundation, how would you implement a message passing scheme that:

a) Always waits for the receiver to receive the message before the sender continues

b) Allows messages greater than 10 KB.

c) Has unlimited buffering so that a sender never has to wait

d) Won't lose a message if the receiver dies

e) Has unlimited buffering?

Segment Review

You should be able to:

- Explain the concept of a process
- Describe the costs associated with a process switch and a thread switch
- Contrast processes and threads
- Determine necessary fields for a process control block
- Describe the possible scheduling states for a process for both two-state and five-state process models
- Categorize message passing systems
- Explain how to implement one interprocess communications mechanism in terms of another