CS 214 Recitation(Sec. 6)

Zuohui Fu
Ph.D. Department of Computer Science
Office hour: Mon 2pm-3pm
Email: zf87 AT cs dot rutgers dot edu

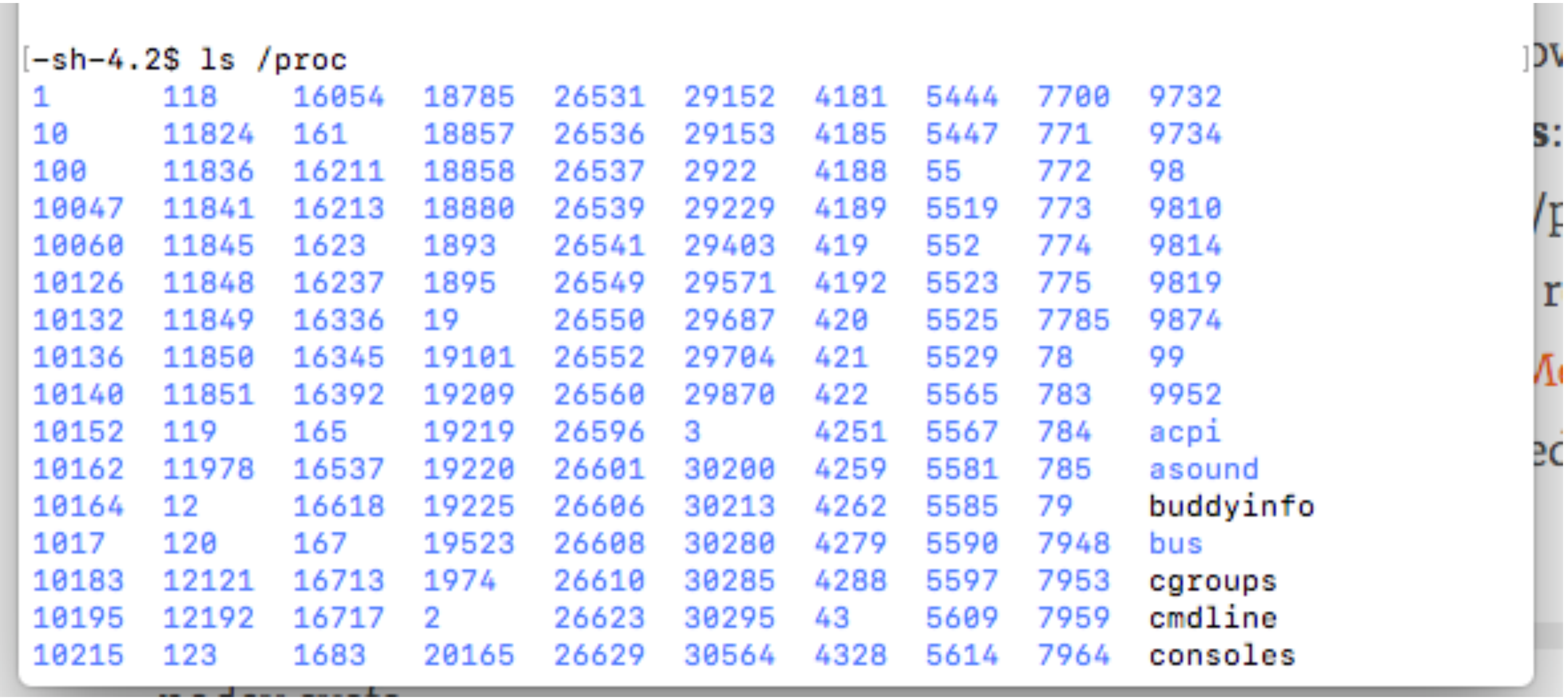
11/02/2017

Topics

- Solution of HW 5
- Thread and Process
- Thread creation and joining

Solition-5-process information

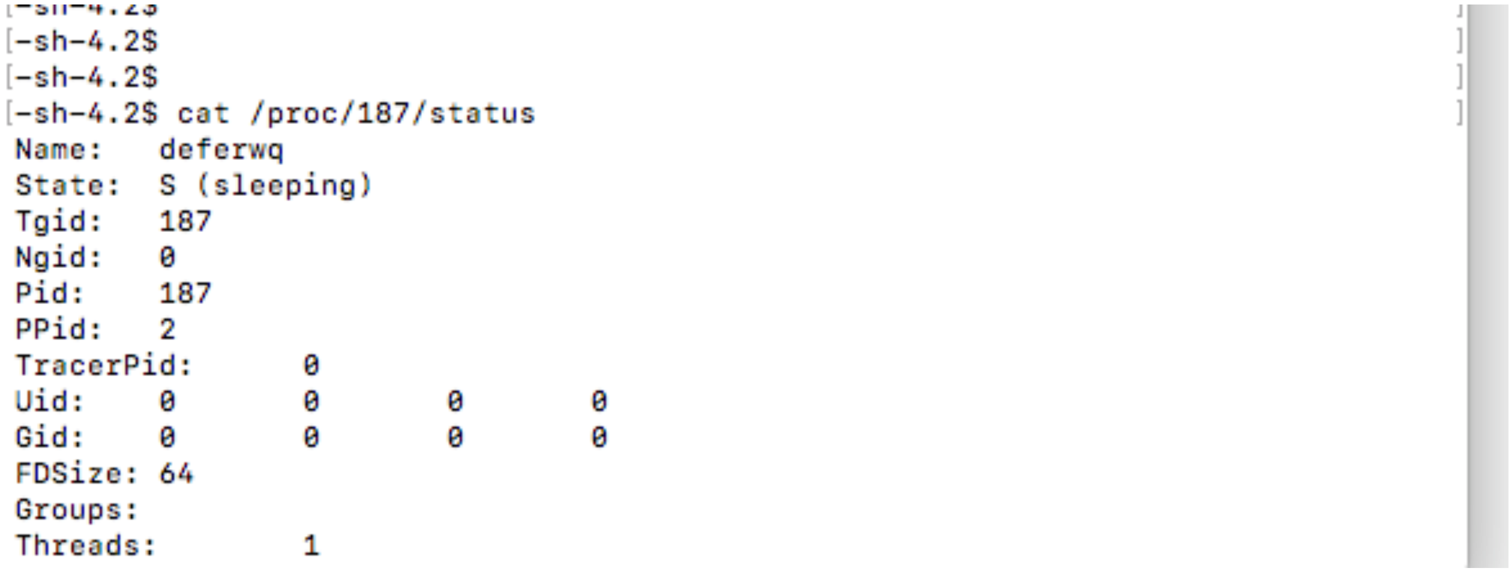
- How to get process information
  - The information for eaxh process is stored in a folder name by its PID.



```
[-sh-4.2$ ls /proc
1      118    16054  18785  26531  29152  4181  5444  7700  9732
10     11824  161    18857  26536  29153  4185  5447  771   9734
100    11836  16211  18858  26537  2922   4188  55    772   98
10047  11841  16213  18880  26539  29229  4189  5519  773   9810
10060  11845  1623   1893   26541  29403  419   552   774   9814
10126  11848  16237  1895   26549  29571  4192  5523  775   9819
10132  11849  16336  19     26550  29687  420   5525  7785  9874
10136  11850  16345  19101  26552  29704  421   5529  78    99
10140  11851  16392  19209  26560  29870  422   5565  783   9952
10152  119    165    19219  26596  3      4251  5567  784   acpi
10162  11978  16537  19220  26601  30200  4259  5581  785   asound
10164  12     16618  19225  26606  30213  4262  5585  79    buddyinfo
1017   120    167    19523  26608  30280  4279  5590  7948  bus
10183  12121  16713  1974   26610  30285  4288  5597  7953  cgroups
10195  12192  16717  2      26623  30295  43    5609  7959  cmdline
10215  123    1683   20165  26629  30564  4328  5614  7964  consoles
```

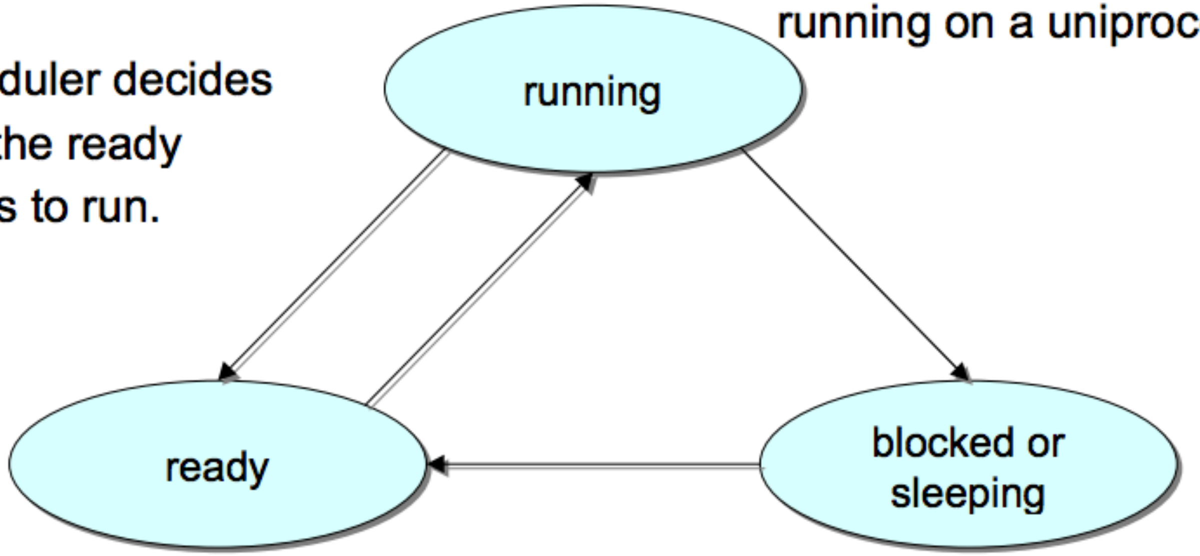
Solition-5-process information

For more, please visit:
https://www.linux.com/news/discover-possibilities-proc-directory
http://man7.org/linux/man-pages/man5/proc.5.html
Ex: file 'status' (comm, schedstat)



```
[-sh-4.2$
[-sh-4.2$
[-sh-4.2$
[-sh-4.2$ cat /proc/187/status
Name:   deferwq
State:  S (sleeping)
Tgid:   187
Ngid:   0
Pid:    187
PPid:   2
TracerPid:      0
Uid:    0       0       0       0
Gid:    0       0       0       0
FDSize: 64
Groups:
Threads:        1
```

Process



Only one process can be running on a uniprocessor

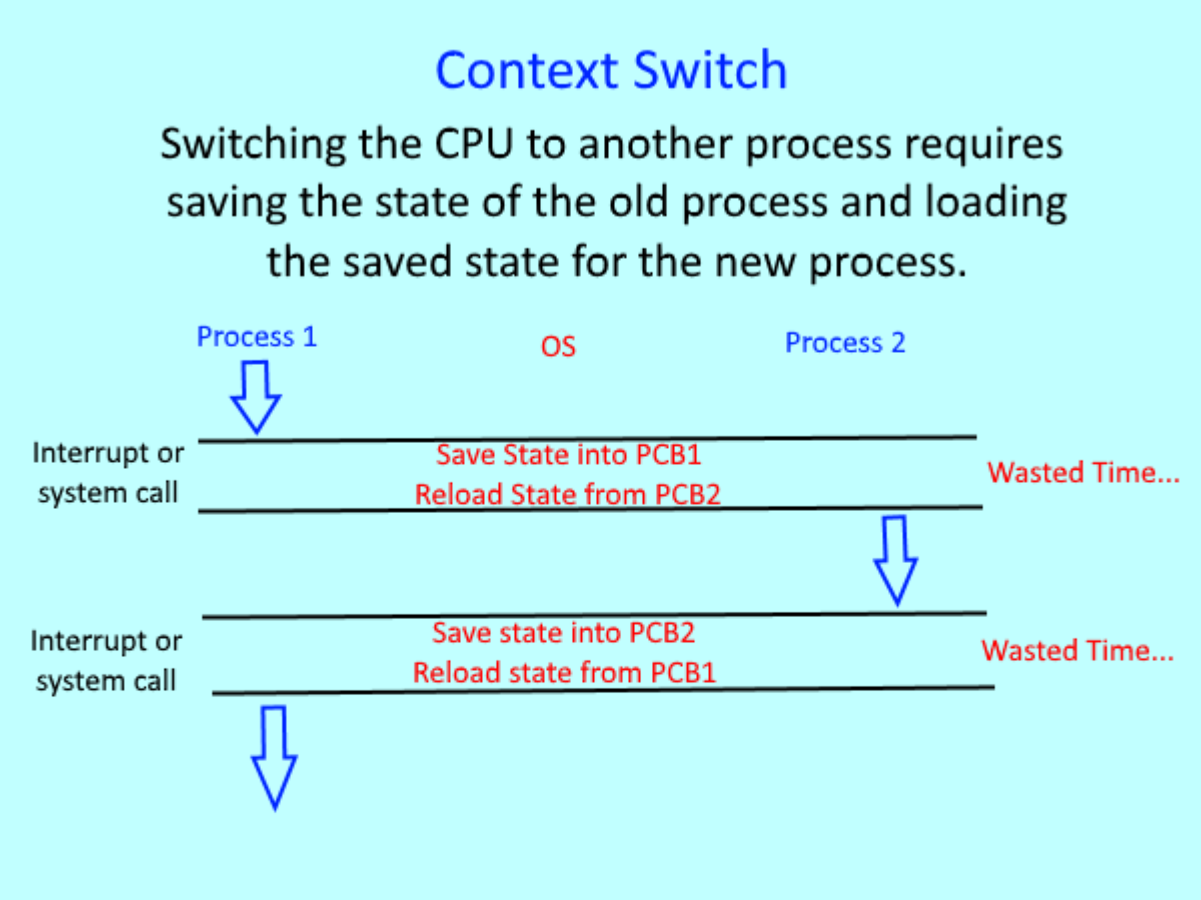
The scheduler decides which of the ready processes to run.

A process is ready if it could use the CPU immediately.

A process is blocked if it waiting for an event (I/O, signal)

Context Switch

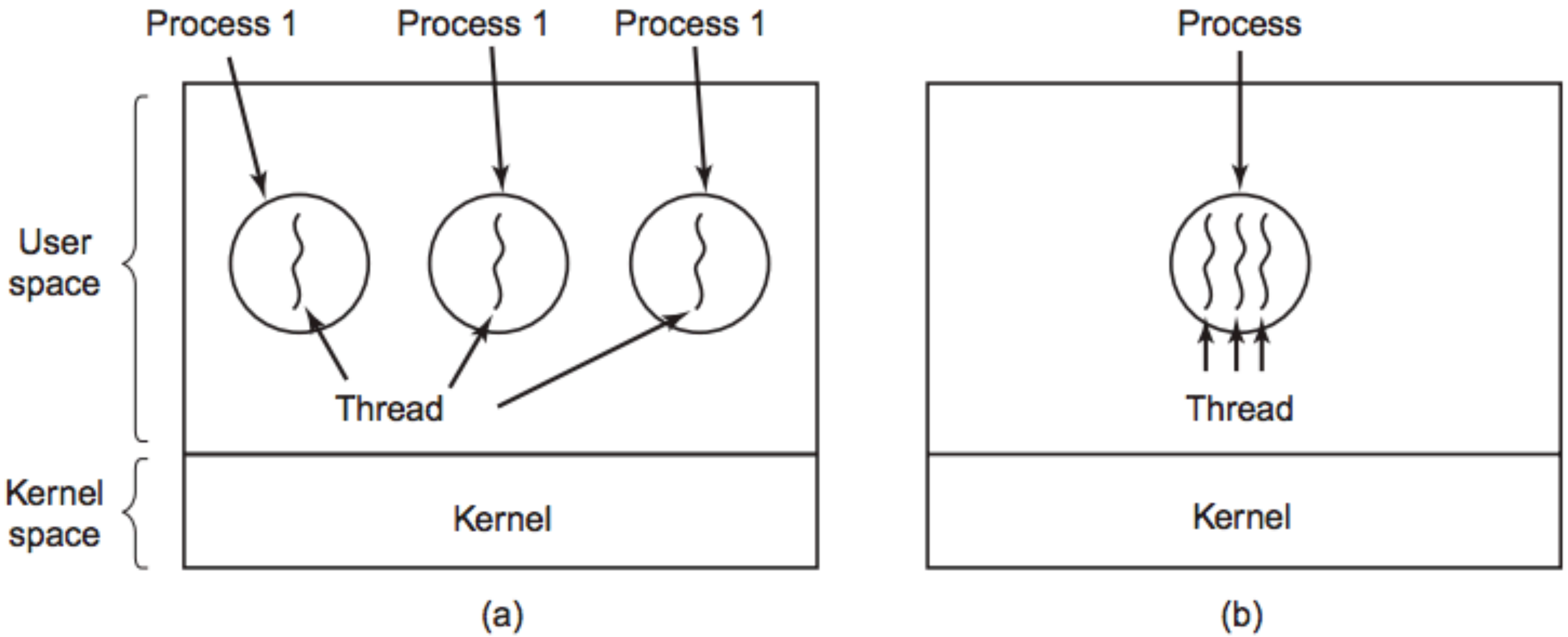
A context switch is the computing process of storing and restoring the state (context) of a CPU such that multiple processes can share a single CPU resource. The context switch is an essential feature of a multitasking operating system.



Context Switch

Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process.

Properties of threads



(a) would be used when the three processes are essentially unrelated, whereas(b) would be appropriate when the three threads are actually part of the same job and are actively and closely cooperating with each other.

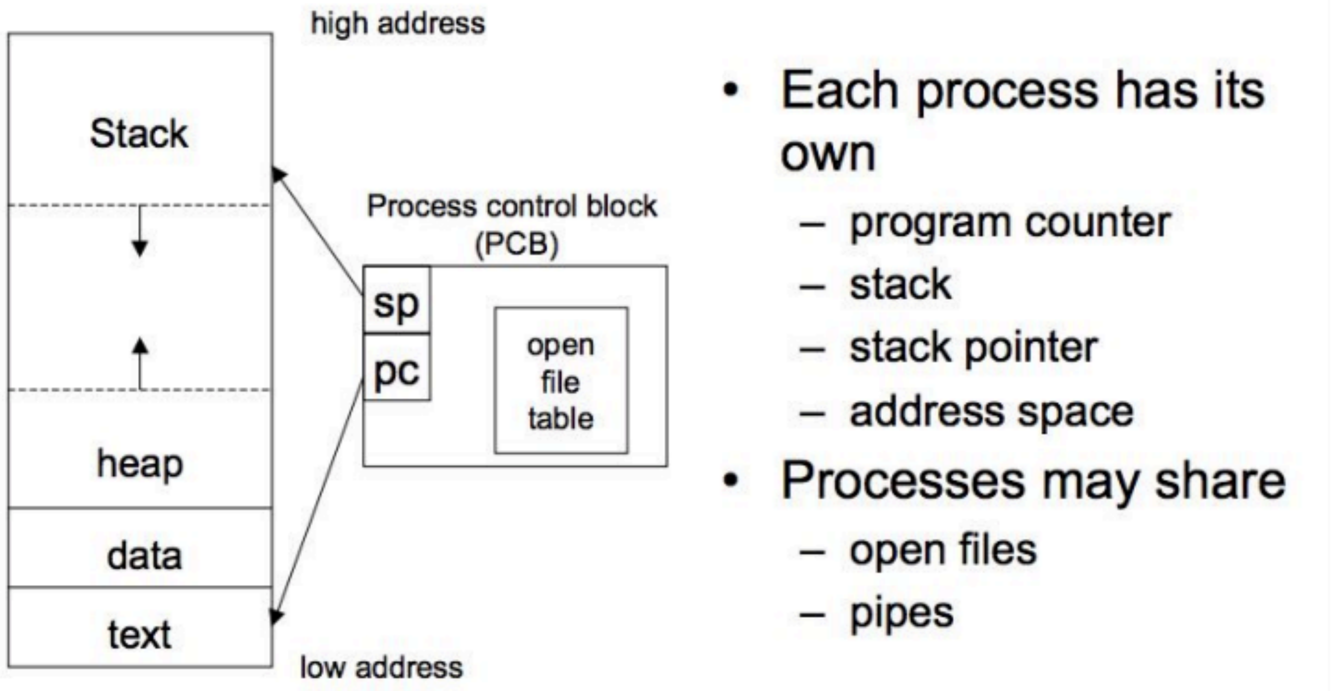
PCB

PCBs are data structures:
– dynamically allocated inside OS memory

When a process is created:
– OS allocates a PCB for it
– OS initializes PCB
– OS puts PCB on the correct queue

As a process computes:
– OS moves its PCB from queue to queue

When a process is terminated:
– PCB may hang around for a while (exit code, etc.)
– eventually, OS deallocates the PCB

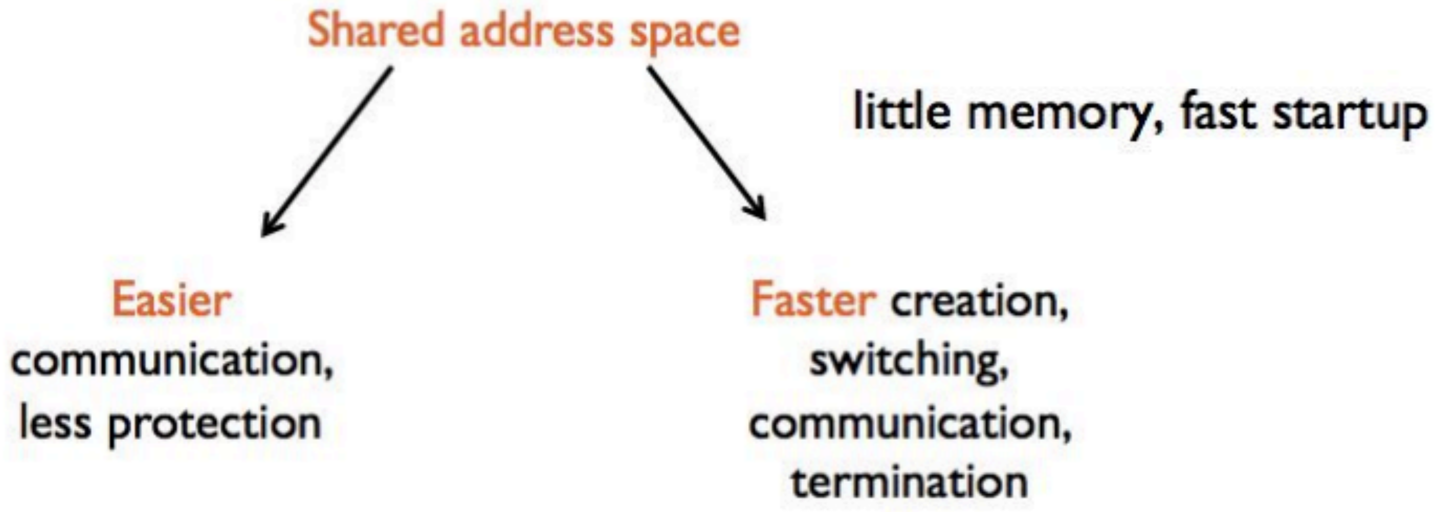


- Each process has its own
  - program counter
  - stack
  - stack pointer
  - address space
- Processes may share
  - open files
  - pipes

https://courses.cs.washington.edu/courses/cse451/05wi/lectures/4-process.pdf

Thread

- Light-weight processes



Shared address space

little memory, fast startup

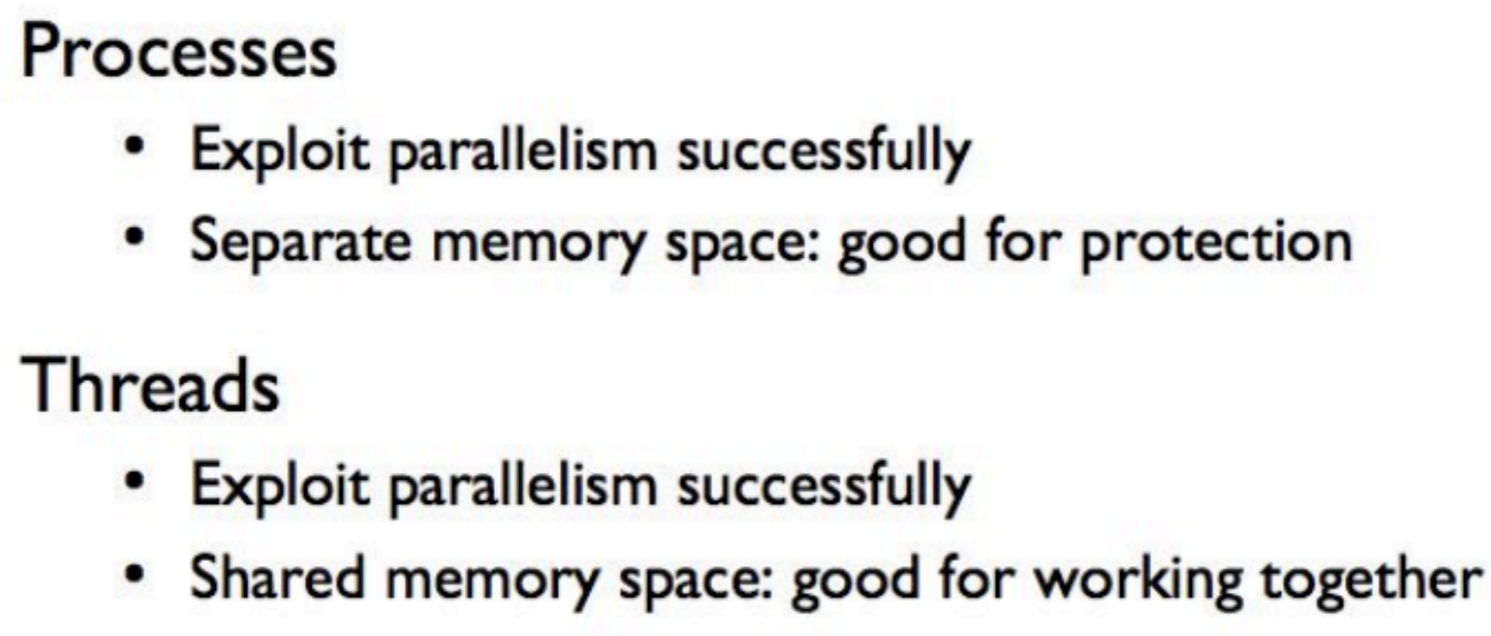
Easier communication, less protection

Faster creation, switching, communication, termination

Thread-Propersites

- Threads
- Execute in same address space
  - separate execution stack, share access to code and (global) data
- Smaller creation and context-switch time
- Can exploit fine-grain concurrency
- Easier to write programs that use asynchronous I/O or communication

Process vs threads



Processes

- Exploit parallelism successfully
- Separate memory space: good for protection

Threads

- Exploit parallelism successfully
- Shared memory space: good for working together

Thread-continue

- User-level vs kernel-level threads
  - kernel not aware of threads created by user level thread package (e.g. Pthreads), language (e.g. Java)

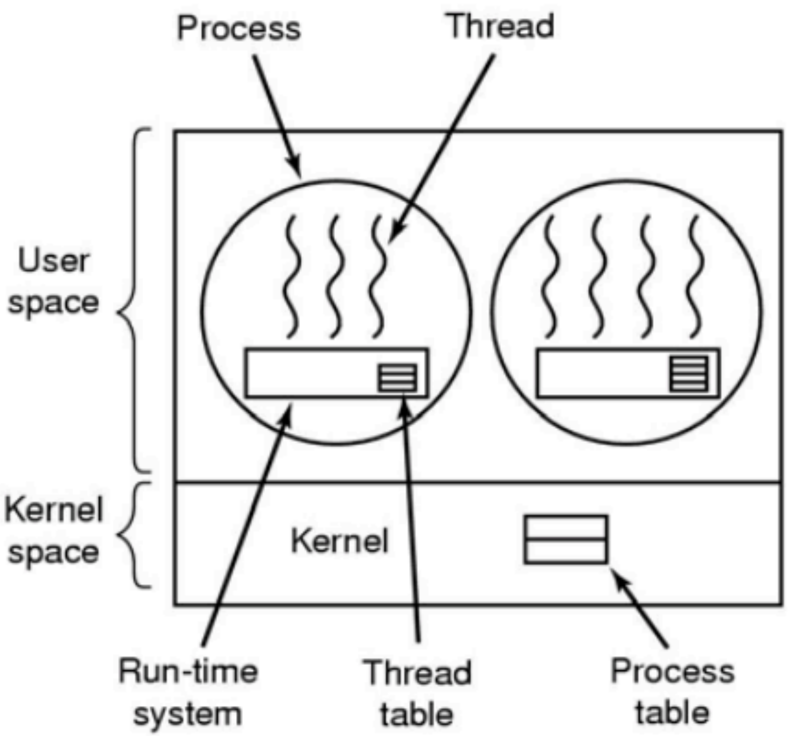
  - user-level threads typically multiplexed on top of kernel level threads in a user-transparent fashion

Implementing Threads in User Space

Threads managed by a threads library
– Kernel is unaware of presence of threads

Advantages:
– No kernel modifications needed to support threads
– Efficient: creation/deletion/switches don’t need system calls
– Flexibility in scheduling: library can use different scheduling algorithms, can be application dependent

Disadvantages
– Need to avoid blocking system calls
– Threads compete for one another
– Does not take advantage of multiprocessors [no real parallelism]

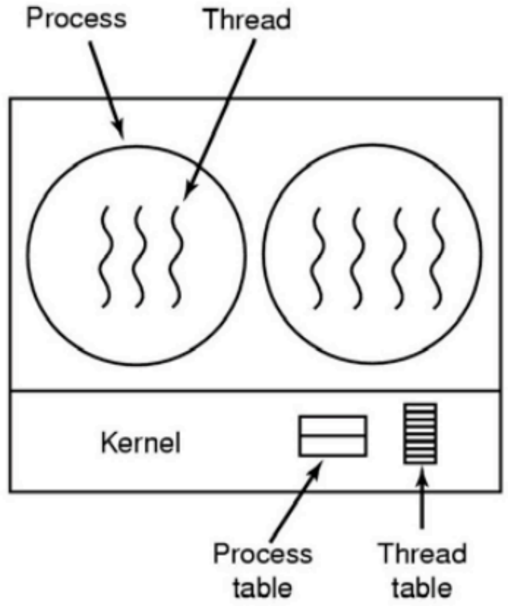


A user-level threads package

Kernel Level

- Shared virtual address space
- Contains running state data
- Less overhead
- From the OS's point of view, this is what is scheduled to run on a CPU

- No need to create a new address space
- No need to change address space in context switch
- Kernel aware
- Still need to enter kernel to context switch



A threads package managed by the kernel

Thread Creation and Joining

- A thread can be terminated by
  - Returning from the thread function
  - the ***main()*** function exiting or exit() called or sending a ***SIGTERM*** signal
  - ***pthread_exit*** - join with a terminated thread
  - ***pthread_cancel*** - send a cancellation request to a thread

- What is the difference between ***exit*** and ***pthread_exit***?
  - exits(): exits the entire process and sets the processes exit value. All threads inside the process are stopped
  - pthread_exit(void*): only stops the calling thread. The pthread library will automatically finish the process if there are no other threads running.

- Passing Arguments to Threads:

  http://www.cs.toronto.edu/~krueger/csc209h/lectures/Week13-threads-4.pdf