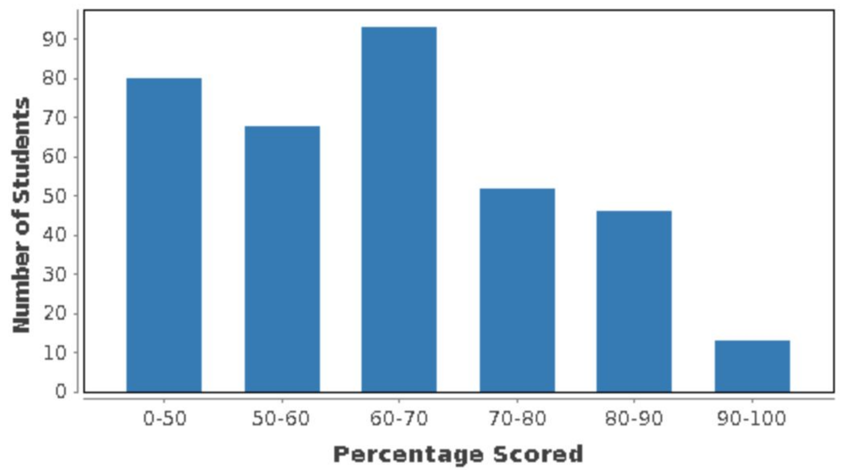
Recitation 9

TA Hanxiong Chen
hc691@rutgers.edu

Midterm statistics

Grade Statistics for Midterm 0



Average (Mean) Sco...	66.72 / 110 (60.66%)
Median Score	68 / 110 (61.82%)
Standard Deviation	18.68
Lowest Score	3 / 110 (2.73%)
Highest Score	109 / 110 (99.09%)
Total Graded Scores	352

Process

Process Control Block (PCB)

- A structure in the operating system representing the processes. It defines the state of OS.
- Process table: contains all the current processes in the operating system.

Process has its own memory space. To create a new process need amount of overhead.

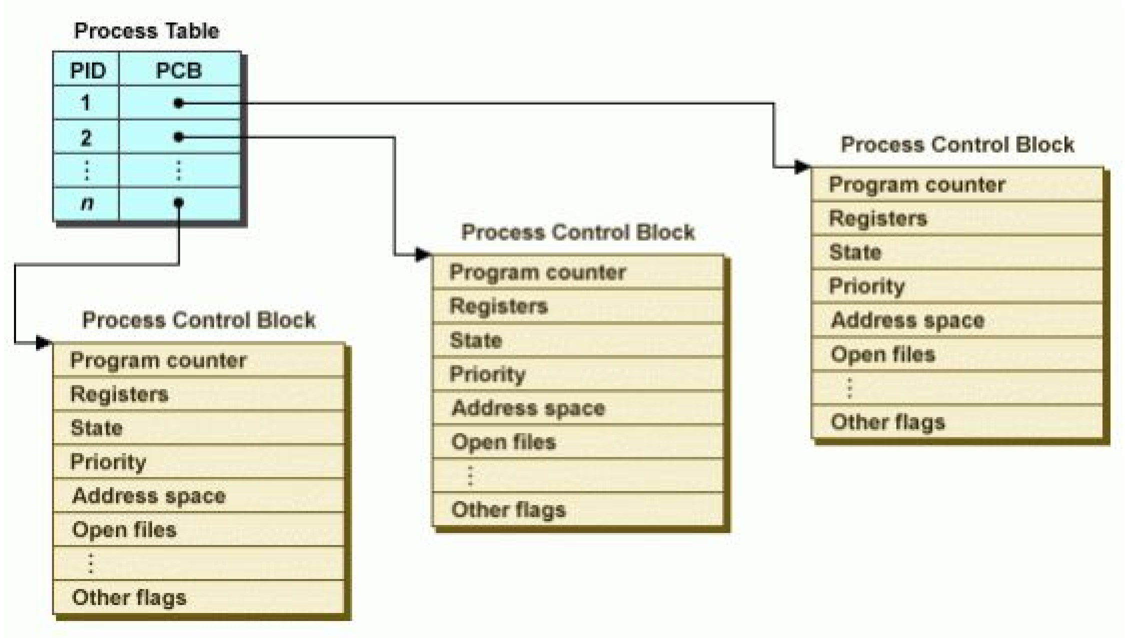
Process (cont’d)

PCB includes:

- Process ID
- Process State (state diagram)
- Registers (and program counter)
- Memory info
- List of open files, inter process communication info
- Accounting info.
- Pointers to other data structures in the OS

Process (cont'd)

PCB Table:



Looks the same as metadata of malloc or inode table?

Thread

- Thread is a light-weight processs.
  - Fork and pthread_create() both call clone()
  - Every process has at least one thread (main thread) tricky questions...
- What stuff do threads share and what not?
  - Text segment, data segment, bss segment, heap…
  - They share almost everything except for stack
  - Each thread has its own stack frame, stack pointer and program counter

Thread vs Process

The major differences between threads and processes are:

1. Threads share the address space of the process that created it (each thread has its own call stack); processes have their own address space.
2. Threads have direct access to the data segment of its process; processes have their own copy of the data segment of the parent process.
3. Threads can directly communicate with other threads of its process; processes must use interprocess communication to communicate with sibling processes.
4. New threads are easily created; new processes require duplication of the parent process.
5. Threads can exercise considerable control over threads of the same process; processes can only exercise control over child processes.

Kernel-level Thread vs User-level Thread

User-level Thread:

- Switch between user-level threads are faster than kernel-level threads. (no need to reset memory protection to switch to kernel scheduler and then switch back)
- User-level threads require less kernel support, which can make the kernel simpler. But only one thread is allowed to run at one time!

Kernel-level Thread:

- Kernel-level threads allow a thread to run while another thread in the same process is blocked
- Kernel-level threads can run simultaneously on multiprocessor machines, which purely user-level threads cannot achieve.

Is Multi-Thread always good?

Think about this question.

Please write a program to count number from 1 to 100 with 10 threads.

IS THIS GOOD TO DO MULTI-THREAD?

Multi-Thread

Advantages

- Make use of wasted waiting time. (especially for I/O)
- Sharing the cache. Make use of the system resources efficiently.
- Easy to work together. (communicate with each other)

Disadvantages

- Switching contexts need time consumption.
- Operating on shared data may cause synchronization issues.

To make multi-thread useful, scheduling is very important

Question

Why use threads over processes, or why use processes over threads?

- Time consumption of creating a new thread/process
- Do you need do communication between threads/processes frequently?
- Do you need to have a independent memory space?

Multi-thread (cont’d)

Create a thread using pthread library

1. Create a ‘pthread_t’ type variable (tcb).
2. Use pthread_create() to create a new thread
3. Use pthread_join() if you want your current thread to wait for other threads.
4. Use pthread_detach() if you don’t need to wait for any threads.
5. Use pthread_exit() to terminate your thread
6. Use -lpthread to link library when compiling

Pthread

- pthread_join()
  - Wait for the thread to finish.
  - Clean up any resources associated with the thread.
- pthread_detach()
  - Release resource when the thread is done
  - No other thread can join on a detached thread

If a thread is terminated, such as call pthread_exit(), it will not release the TCB memory space. There are two ways to release the TCB:

1) the thread who join on that thread will pick up the return value and clean the resources;
2) set the thread state as detached.

Thread-safe

- What result in the un-safety in multi-thread?

```
char* to_message(int num) {
  char static result [256];
  if(num < 1000)
    sprintf(result, "%d : blah blah" , num);
  else strcpy(result, "Unknown");
  return result;
}
```

- When we talk about sharing things, synchronization issue comes.
  - asctime(), getenv(), strtok(), strerror() function are not thread-safe… WHY?

If the function try to create/operate on shared data without any protection, this function is not multi-thread safe.

Others

What is the difference between exit() and pthread_exit()?

What is the difference between pthread_cancel and pthread_exit()?

DEMO TIME!!!

Others

What is the difference between exit() and pthread_exit()?

A: exit() will terminate your whole process while pthread_exit() only terminate your thread.

What is the difference between pthread_cancel and pthread_exit()?

A: 1. pthread_exit() can pass return value to the threads who is joining on it, but pthread_cancel() can't;

2. pthread_cancel() can terminate other threads but pthread_exit() can't.

3. If you just want to terminate a thread itself without returning anything, then they are the same.

Next time: Synchronization

Reference

1. "Parallel and Distributed Programming Using C++" by Cameron Hughes, Tracey Hughes, Table 4-1
2. http://cs.stackexchange.com/questions/1065/what-is-the-difference-between-user-level-threads-and-kernel-level-threads
3. http://softwareengineering.stackexchange.com/questions/97615/what-can-multiple-threads-do-that-a-single-thread-cannot
4. http://www.thegeekstuff.com/2012/04/create-threads-in-linux/?utm_source=feedburner
5. http://stackoverflow.com/questions/22427007/difference-between-pthread-exit-pthread-join-and-pthread-detach
6. http://www.perlmonks.org/?node_id=306063
7. http://www.thegeekstuff.com/2012/04/terminate-c-thread/