CMPSCI 377 Operating Systems **Fall 2005**

Lecture 4: September 22

Lecturer: Emery Berger *Scribe: Andrey Ostapchenko and Nikita Ivanov*

Today:

- OS Structures and Processes

4.1 Processes

OS executes variety of programs:

Batch System jobs: run one after the other
Time-Shared systems: user programs or tasks

Definition 4.1 *A process is a fundamental schedulable unit of execution*

At minimum, each process includes:

- Program counter (stack location)
- Stack (variable values)
- Data section (global variables, address space)

Process states:

New: being created, i.e. process goes into memory from the disk
Running: instructions being executed
Waiting: waiting for some event to occur, i.e. awaiting data from disk
Ready: waiting to be assigned to a processor
Terminated: self-explanatory
Zombie: useless children processes spawned by a parent process that has crashed

Processes usually loop between 3 states - running, waiting, ready. Transitions from one state to another usually happen on program actions, OS actions, or interrupts

As an example, in UNIX a user can choose to run the process and wait until it finishes, or run it in background:
Sequential run: cp -r foo /bar
That will recursively copy foo into bar and then prompt for another command.
Background run: cp -r foo /bar &
That will run the same process, but user will be able to continue working while the process is running.

4.1.1 Process Control Block

Definition 4.2 *A **Process Control Block** or PCB tracks process state, such as program counter, cpu registers, I/O status, memory-management*

As an example of a poorly designed PCB, consider the following: in Linux, process identifiers are limited to 1024. If the number of processes increases past that number, the operating system will crash.

Note that it is expensive to switch between processes. When a user switches from one process to another:
TLB needs to be flushed and repopulated
Memory cache is useless ("cold"), so it needs to be renewed.

The OS usually tries to solve this problem by scheduling each process to run for a specific period of time. This job is handled by the scheduler using queues.
Scheduler queues:

Job queue: the set of all processes in a system
Ready queue: set of processes residing in main memory ready and waiting to be executed
Device queue: set of processes waiting for I/O device - one per device

These queues are simple linked lists. Same item can be placed on more than one queue at the same time.

4.2 Forking

It is possible for one process to create another process - this is called **forking**. The creator is called the **parent process** and the new processes are called **children processes**. A parent process can wait for the children processes to complete or it can continue running in parallel.

UNIX: $fork()$ copies variables/registers from parent to child.
Memory lazily copied - copy-on-write (by reference)
The only difference between parent and child is the return value - parent returns pid (process ID) of child; child returns 0.

On termination of a process the operating system reclaims all resources.

UNIX processes have the ability to terminate themselves via the *exit* system call; they can also terminate child processes via the *kill* system call.

Processes can also *cooperate* and work together on a task. This may improve performance and yield possible simpler program design.

4.3 Process Communication

Processes sometimes need to exchange information between themselves. This can be done in two ways:

- **Message passing:** send/receive information via sockets, pipes
- **Shared memory:** establish mapping to named memory object
  - use *mmap*
  - $fork()$ processes to share the structure

4.4 Process: a unit of execution

Process is represented by **Process Control Blocks**. They contain process state, scheduling info etc. A uniprocessor system can only have one running process at a time, so context switch is used to alternate between processes.

Definition 4.3 ***Quantum*** *is the maximum time the process is allowed to run by the scheduler before it gets placed back in queue (or finishes and terminates).*

Since a process can only run for a limited time (quantum), execution time needs to be divided into quanta for every process. That is called **Time Slicing**.