

Classic Processes (CSE 422S)

Ken Wong
Washington University

kenw@wustl.edu
www.arl.wustl.edu/~kenw

fork, exec and the shell (1)

- Example: Interactive Commands

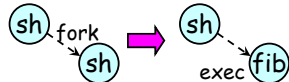

```
> fib 10 > fib10.out
> fib 2000 > fib2000.out &
```
- Your interactive shell (tcsh, bash) interprets commands

Run in the background
- Shell processing of 'fib 10'
 - » Shell looks for the 'fib' executable in a directory listed in the PATH environment variable
 - e.g., ".:usr/home/kenw/bin:/usr/bin:/usr/local/bin"
 - » Forks a copy of the shell; i.e., creates a child process
 - » Child redirects stdout to the file 'fib10.out'
 - » Child execs 'fib' passing in command-line arguments
 - 'fib' process replaces the shell's child process
 - » Parent shell waits for 'fib' process to terminate
 - » Display prompt after 'fib' terminates

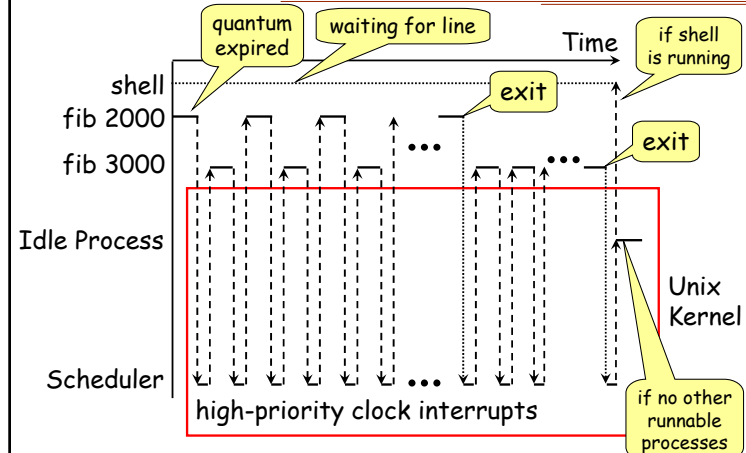
fork, exec and the shell (2)

- Shell processing of 'fib 2000'
 - » Processing is the same as for 'fib 10' except that parent does not wait for child to terminate
- Extended Example:


```
> fib 10 > fib10.out
> fib 2000 > fib2000.out &
> fib 3000 > fib3000.out &
...
> wait $! # last bg proc
```
- Fundamental Abstraction: The Process
 - » *Traditional Process*: The process executes a single sequence of instructions in an address space
 - » The program counter (PC) tracks the sequence of instructions
 - » *Modern Process*: Multiple control paths

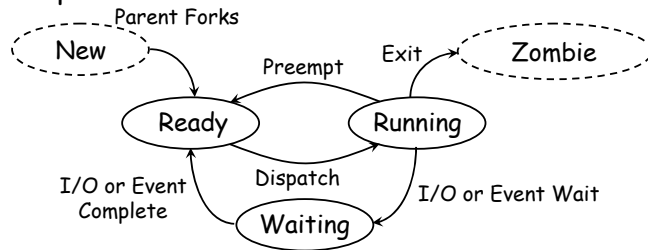


Two CPU Intensive Processes



Process Run States (Simplified)

Basic process run states



Other possible states

- » Stopped: Not terminated, but not to be scheduled
- » Zombie (to be described)
- » Running: In user space or kernel space

5 - Ken Wong, 2/5/2007

Washington University in St. Louis

Clocks

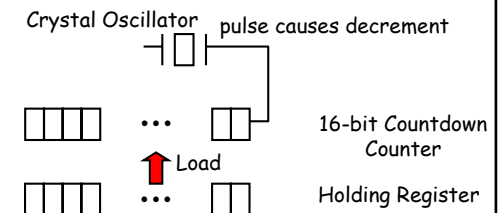
Simple Clock

- » Interrupts on every voltage cycle (50 or 60 Hz)

More Typical Clock (Programmable)

- » Frequencies above 1 GHz (1 nsec per pulse)
- » Every pulse causes counter to decrement
- » Interrupt when counter reaches 0

- » Program to interrupt every **10 msec tick**



6 - Ken Wong, 2/5/2007

Washington University in St. Louis

Clock Software (1)

- Hard Clock routine is called after each 10 msec tick
- Keep time of day
 - » Record time since Jan 1, 1970 during system boot
 - » Count ticks since system boot
 - » Compute actual time/date when user asks for time/date
- Prevent processes from getting too much CPU time
 - » Decrement a process' quantum counter after every tick
 - » Clock driver calls CPU scheduler when quantum reaches 0
- Account for CPU usage
 - » Update process' usage after each tick (inaccurate)

7 - Ken Wong, 2/5/2007

Washington University in St. Louis

Clock Software (2)

- Handle alarm and other "timer" system calls
 - » Call `softclock()` routine if current interrupt-priority level is low enough
 - » `softclock()` handles lower-priority timer processing
 - Run at software-interrupt level
 - e.g., TCP timer tick is 200 msec or 500 msec
- Watchdog timers
 - » e.g., resume spinning floppy disk
- Profiling, monitoring, statistics gathering
 - » Process region usage histogram

8 - Ken Wong, 2/5/2007

Washington University in St. Louis

Context Switching

■ Definition

- » The activity that occurs when the OS kernel switches between processes in an effort to share the CPU among competing, runnable processes

■ Actions

- » Save contents of hardware registers (PC, SP, ...)
- » Load PC with location of resume point
- » Load hardware registers with new context if full context switch

■ Context-switch time is overhead

- » CPU utilization affected by quantum and context-switch time
 - *Quantum*: *Time-slice* (max CPU interval) given to a process

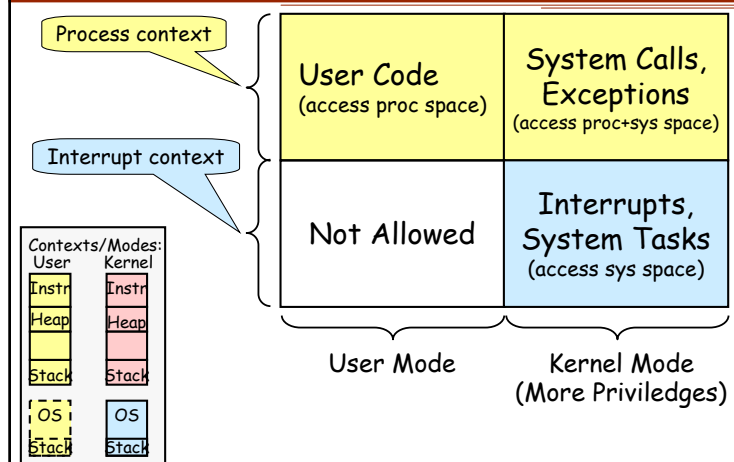
■ Types

- » *Voluntary*: Process blocks
- » *Involuntary*: End of time quantum or higher-priority, runnable process gets control of CPU

9 - Ken Wong, 2/5/2007

Washington University in St. Louis

Execution Context Versus Mode (1)



10 - Ken Wong, 2/5/2007

Washington University in St. Louis

Execution Context Versus Mode (2)

■ Execution Mode

» User Mode

- Some parts of virtual address space can not be accessed
- Some instructions (e.g., memory management) can not be executed

» Kernel Mode

- Can access kernel address space
 - Is a fixed part of the virtual address space of every process
- System call puts user into kernel mode

■ Unix kernel is reentrant

- » *Reentrant*: Access to shared data must be coordinated
- » Multiple processes can be "in the kernel" → Each process needs its own kernel stack

■ Execution Context

- » *Process context*: Kernel acts on behalf of user process
- » *Interrupt context*: Kernel can't access context of current process

11 - Ken Wong, 2/5/2007

Washington University in St. Louis

Process Control Block (PCB)

■ Process Management

- » Registers (PC, SP, GRs)
- » Program Status Word (PSW)
- » Scheduling (priority)
- » PID, PPID
- » Usage accounting
- » Signals, Timers

■ Memory Management

- » Pointers to text, data, and stack segments
- » Page table

■ File Management

- » Directories (current, root)
- » File descriptors
- » UID, GID

12 - Ken Wong, 2/5/2007

Washington University in St. Louis

Signal Sources

- **Command Line**
 - » e.g., "kill -STOP 1899"
 - Send SIGSTOP signal (19) to process 1899
- **Process**
 - » e.g., "rc = kill(1899, SIGSTOP);"
- **Keyboard (generated by terminal driver)**
 - » **Interrupt key (ctrl-c)**
 - Generates SIGINT; terminate process
 - Signal sent to all processes in foreground process group
 - » **Quit key (ctrl- \backslash)**
 - Generates SIGQUIT; terminate process with core dump
 - » **Stop key (ctrl-z)**
 - Generates SIGSTP; stop process (can be resumed)
 - SIGSTP (interactive stop) is different from SIGSTOP
 - Signal sent to all processes in foreground process group

13 - Ken Wong, 2/5/2007

Washington University in St. Louis

Unix Process Hierarchies

- **Process Group**
 - » A process and all of its descendants which have not changed the default process group ID (via *setpgid(2)*)
- **User sends signal from keyboard**
 - » Signal delivered to all members of process group associated with keyboard
 - » Each process can:
 - Catch signal
 - Ignore signal
 - Take default action (terminate)
- **The init process: root parent of all user processes**
- **Windows doesn't enforce a hierarchy**

14 - Ken Wong, 2/5/2007

Washington University in St. Louis

Separate fork from exec

- **Separating out fork from exec**
 - » Greater flexibility
 - » Simplifies implementation
- **Potential operations before 'exec'**
 - » Redirect stdin, stdout, or stderr
 - » Close open files inherited from parent
 - » Change UID or process group
 - » Reset signal handlers
- **Can get effect of copying parent without all of cost**
 - » Copy-on-write:
 - Child shares virtual memory pages with parent but is read-only
 - A write to a shared page causes the actual page to be copied
 - » vfork:
 - Child temporarily uses parent's memory pages (dangerous!)

15 - Ken Wong, 2/5/2007

Washington University in St. Louis

Unix Process Creation (fork)

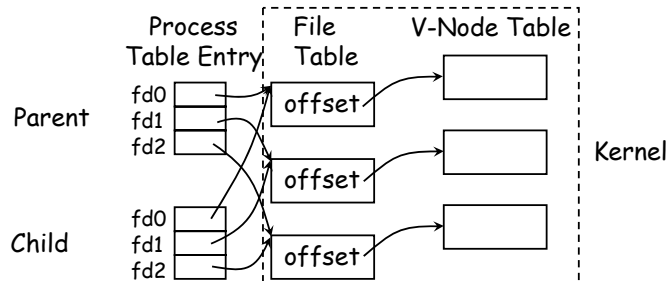
- **Reserve swap space for child's data and stack**
- **Allocate new PID and proc structure**
- **Init child's process control block (PCB)**
 - » Copy UID, GID, and signal masks from parent
 - » Reset (zero) statistics
 - » Copy parent's registers to child's hardware context
- **Setup virtual memory tables**
 - » Child's data pages are the same as parent's except are read-only
- **Mark child runnable and give to scheduler**
- **Return PID = 0 to child and child PID to parent**

16 - Ken Wong, 2/5/2007

Washington University in St. Louis

Effect of fork

- Child gets a copy of its parent's memory
 - » BUT changes made by child are not reflected in parent ...
 - » EXCEPT a child can affect the I/O state of parent
 - File descriptors can point to same file table entry
 - » WARNING: Parent should "fflush(stdout)" before fork



17 -Ken Wong, 2/5/2007

Washington University in St.Louis

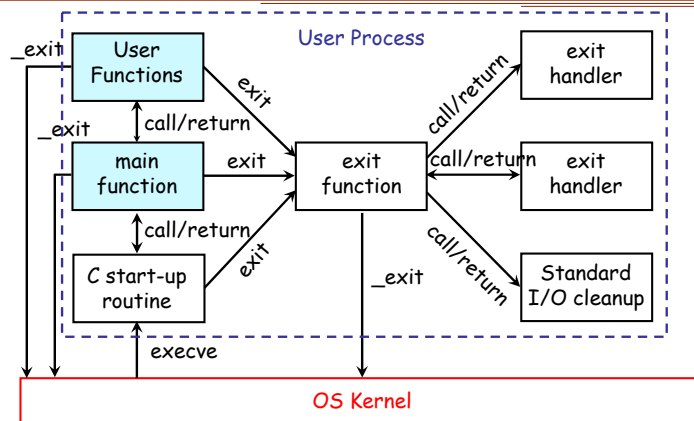
Exec Operations

- Find executable file (argv[0])
- Verify that user may exec file
- Check that file header is a valid executable
- If file is *setuid* or *setgid*, change caller's effective UID and/or GID
- Copy exec args and env variables to kernel space
 - » Current user space will be destroyed
- Allocate heap and stack and free old process space
- Setup new address space
 - » Copy exec args and env variables into new space
- Reset all signal handlers
- Initialize hardware context

18 -Ken Wong, 2/5/2007

Washington University in St.Louis

Unix C Program Startup/Termination



19 -Ken Wong, 2/5/2007

Washington University in St.Louis

Process Termination

- Termination Condition
 - » Normal exit (see exit(3) and _exit(2))
 - » Error exit
 - » Fatal error (e.g., illegal instr, bad addr, divide by zero)
 - » Killed by another process (*kill* system call)
- Normal Unix Process Termination Activity
 - » Close all files
 - » Save usage stats
 - » Make init process the parent of live children
 - » Change run state to ZOMBIE
 - » Release memory
 - » Send SIGCHLD to parent
 - » Wake up parent if asleep
 - » Call the scheduler

20 -Ken Wong, 2/5/2007

Washington University in St.Louis

Unix Zombie Processes

■ A Zombie Process

- » A process that has terminated but whose parent has not waited for it
- » When a process terminates, the OS kernel:
 - Discards all memory used by the process; closes all process' files
 - Keeps some info (PID, exit status, CPU time usage)
 - Provides info to parent when parent calls wait

■ ps output example

UID	PID	PPID	STAT	TT	TIME	COMMAND
7001	5608	5606	Ss	p3	0:03.88	tcsh
7001	7317	5608	S+	p3	0:00.02	testzombie
7001	7318	7317	Z+	p3	0:00.00	testzombie

```
if ((cpid = fork()) == 0)      exit(0);
else { sleep(2);
      system("ps -l");
} // testzombie code
```

init process should harvest zombies

21 - Ken Wong, 2/5/2007

Washington University in St. Louis

waitpid

■ pid_t waitpid(pid_t pid, int *status, int options);

- » Unlike wait(2), waitpid doesn't have to block
 - wait(2) blocks until one of its children terminates

■ pid values

- » pid == -1: Wait for any child to terminate
- » pid > 0: Wait for child whose PID equals pid
- » pid == 0: Wait for any child whose PGID equals the PGID of process pid
- » pid < -1: Wait for any child whose PGID = abs(pid)

■ *status

- » Contains exit status, signal number, and flags

■ options

- » Controls semantics of waitpid

22 - Ken Wong, 2/5/2007

Washington University in St. Louis

Examining waitpid 'status'

■ Contains exit status, signal number, and flags

■ WIFEXITED(status)

- » True if child terminated normally
- » Ex: printf("exit = %d\n", WEXITSTATUS(status));

■ WIFSIGNALED(status)

- » True if child terminated and didn't catch signal
- » Ex: printf("signo = %d\n", WTERMSIG(s));

■ WIFSTOPPED(status)

- » True if child is STOPPED
- » Ex: printf("signo = %d\n", WSTOPSIG(s));

23 - Ken Wong, 2/5/2007

Washington University in St. Louis

waitpid options

■ options = 0

- » No special handling

■ options = WNOHANG

- » Don't block if child is not available and return 0
- » Ex: rc = waitpid(pid, &status, WNOHANG);
if (rc == -1) { ... error ... }
else if (rc == 0) { ... no child available ... }
else { ... rc should equal pid ... }

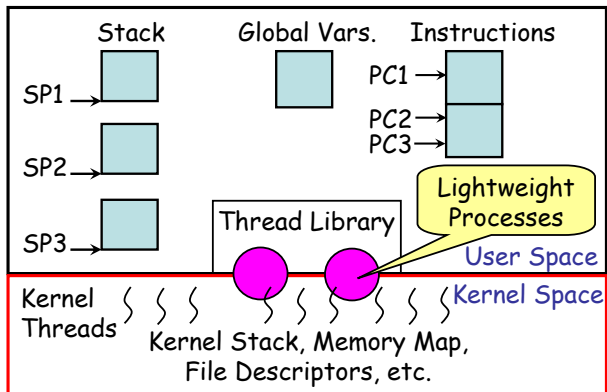
■ options = WUNTRACED

- » Return status of child if stopped child and status has not already been returned (assumes job control support)

24 - Ken Wong, 2/5/2007

Washington University in St. Louis

A Modern Process (1)



25 -Ken Wong, 2/5/2007

Washington University in St. Louis

A Modern Process (2)

- Separate idea of *execution* from *resource grouping*
- Multithreading Support
 - » One or more threads of control (Program Counters)
 - » One stack for each thread of control
- Thread
 - » A unit of local dispatching (scheduling) and has priority
 - » Has a set of CPU registers
 - » Mapped to a lightweight process (LWP)
- LWPs are mapped to processors and globally scheduled
- Global variables are shared by all threads
- System state (file descriptors, working directory, etc) shared by threads

26 -Ken Wong, 2/5/2007

Washington University in St. Louis