

Process Types



• Heavyweight processes

- “Normal” process

• Threads

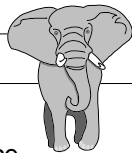
- Multi-processing inside “process”

• Lightweight processes

- Sometimes “kernel threads”

Warning: terminology varies!

Heavyweight Process



• “First class” process

- Unique virtual address space
- Schedule/dispatch by kernel

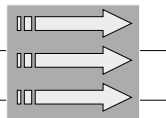
• What’s the problem?

- Large context: expensive switch
- IPC difficult: pipes or ???



More on IPC later

Threads



• Multiple processing streams

- Within a single OS “process”

• Share virtual address space

• Less context (state)

- PC, registers, stack
- Schedule/dispatch by thread code

Thread Applications

- **Background processing**
 - Computation (e.g., spreadsheet)
 - File transfer (e.g., "auto-save")
- **Monitoring asynchronous I/O**
 - Real-time data
- **Manage user interface (GUI?)**

Thread Problems

- **Shared address space**
 - Threads can affect each other's data
- **Not scheduled by kernel (?)**
 - Share resources allocated to process
 - Cannot exploit multiple CPU's
 - Even if no competing processes

Lightweight Processes

- **Hybrid approach**
 - Share virtual address space
 - Scheduled by kernel
- **Intermediate context**
 - Compromise context switch cost
- **Map n threads to m LWP's?**

Sometimes called "kernel threads" (?)

Threads in Windows NT

- **Process**
 - “Container” for threads
 - Address space, object handles, etc.
 - No parent/child relationships
- **Thread**
 - One or more per process
 - Scheduled by kernel

Managing UNIX Processes

- **User job control**
 - Provided by some shells
 - Foreground/background jobs
 - Only foreground gets keyboard input
- **Inter-process control**
 - Parent/child coordination
 - Signals, pipes, shared memory, ...

Process Status

```

torres> ps
  PID TT S      TIME COMMAND
  7178 08 S    0:01.08 -csh (csh)

torres> cat dops
# Use C shell ("x" bit set)
ps

torres> dops
  PID TT S      TIME COMMAND
  7178 08 S    0:01.10 -csh (csh)
  8328 08 S    0:00.07 /bin/csh dops
torres>

```

Background Process

```

torres> cc lab2.c &
[1] 8577
torres> ps
PID TT S      TIME COMMAND
 7178 08 S 0:01.10 -csh (csh)
 8577 08 S 0:00.02 cc lab2.c
 8578 08 U 0:00.10 /.../cc/cfe ...
torres> . . .
[1] +Done          cc lab2.c
torres>

```

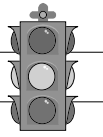
User Job Control

- **Execute job in background**
 - Append “&” to command
 - Control-Z character
 - Stops foreground job
 - Start with fg or bg
- **Display status**
 - ps or jobs command

Background Job Output

- **By default, to cout (stdout)**
 - Same as foreground process
 - May intermix fg/bg output
- **Redirect background output?**
 - E.g., g++ lab2.cpp >lab2_out &
 - Or(?): g++ lab2.cpp >&lab2_out &

UNIX Signals



• Inform process of an event

- Errors: invalid memory reference, ...
- Other: child process termination, ...

• Generated by a process

- Same process: errors, abort()
- kill command, system call

Signal Handling

• Delivered to process

• Caught by handler?

- Set up by process

• If not, abnormal termination?

- Default response to most signals
 - Not some special ones
 - E.g., SIGCONT

Common Signals

SIGHUP	Terminal hang-up	
SIGINT	Interrupt (^C?)	signal.h
SIGILL	Illegal instruction	
SIGABRT	abort()	
SIGFPE	Floating point exception	
SIGSEGV	Memory violation	
SIGKILL	Termination (definite)	
SIGCHLD	Child stop or exit	
SIGALRM	alarm() timeout	

Signal Actions

- **Specified by** `sigaction()`
 - Optionally reports current status
- **Default action** - `SIG_DFL`
- **Ignore** - `SIG_IGN`
- **Program-specified handler**
 - Cannot catch `SIGKILL`

Signal Masks

- **Signals blocked in handler**
- **Type** `sigset_t`
 - `sa_mask` arg to `sigaction()`
 - Arguments to `sigprocmask()`
- **Manipulate with functions**
 - `sigemptyset`, `sigaddset`, ...

Signal Handling Status

- **Ignored**
 - Signal is discarded when received
- **Blocked (signal, not process)**
 - Signal held when received
 - In a pending state
 - Delivered when unblocked