Unix Process Creation

- Parents create children
  - results in a tree of processes
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent’s resources
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate
Unix Process Hierarchies

```
root
  └── pagedaemon
  └── swapper
      └── init
          └── user 1
          ├── user 2
          │   └── user 3
          │       └── ...
          └── user 3
              └── ...
```
Unix Process Creation

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it

- UNIX examples
  - `fork` system call creates new process
  - `exec` family of system calls used after a `fork` to replace the process’ memory space with a new program
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, terminate (default action)

- The `init` process: root parent of all user processes

- Windows doesn’t enforce a hierarchy
Sending Signals to Processes

- **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
Sending Signals to Processes

- Quit key (ctrl-\)
  - 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
    - SIGSTP can be caught or ignored
    - SIGSTOP cannot be caught or ignored.
      - A process receiving this signal will stop until it receives a SIGCONT signal.
      - SIGSTOP is not sendable from a terminal.

- Signal sent to all processes in foreground process group
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 a copy of the parent’s
  - Initially they are the same except are read-only
  - Uses a technique called copy-on-write
Unix Process Creation (fork)

- Mark child runnable and give to scheduler
- Return $\text{PID} = 0$ to child
- Return child PID to parent

Effects 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 “\texttt{fflush(stdout)}” before fork if using printf()
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
exec Operations

- Setup new address space
  - Copy exec args and env variables into new space
- Reset all signal handlers
- Initialize hardware context
exec Functions

- Use to overlay process image with new program
  - Begins executing at its main() function
  - Process ID does not change

- Six exec functions:
  - `execv`
  - `execvp`
  - `execvpe`
  - `execvpl`
  - `execve`
  - `execvp`

- `l` → list of arguments
- `v` → `argv[]` vector
- `P` → file name
- `E` → `environ[]` vector instead of inheriting environment of parent
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)
Process termination

• 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 (usually ignored)
  • Wake up parent if asleep
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 minimal info (PID, exit status, CPU time usage)
  - Provides info to parent when parent calls wait

- Use `ps` command, it reports a snapshot of the current processes
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`)
waitpid

- **status**
  - Contains exit status, signal number, and flags

- **options**
  - Controls semantics of waitpid
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));`
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);`
    ```c
    if (rc == -1) { ... error ... }
    else if (rc == 0) { ... no child available ... }
    else { ... rc should equal pid ... }
    ```
waitpid options

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