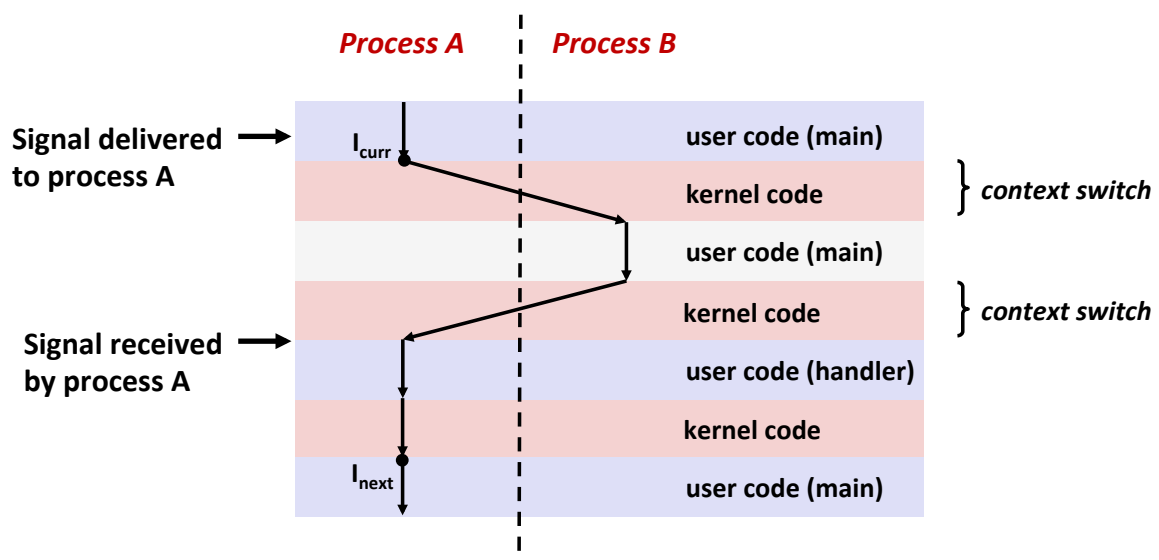


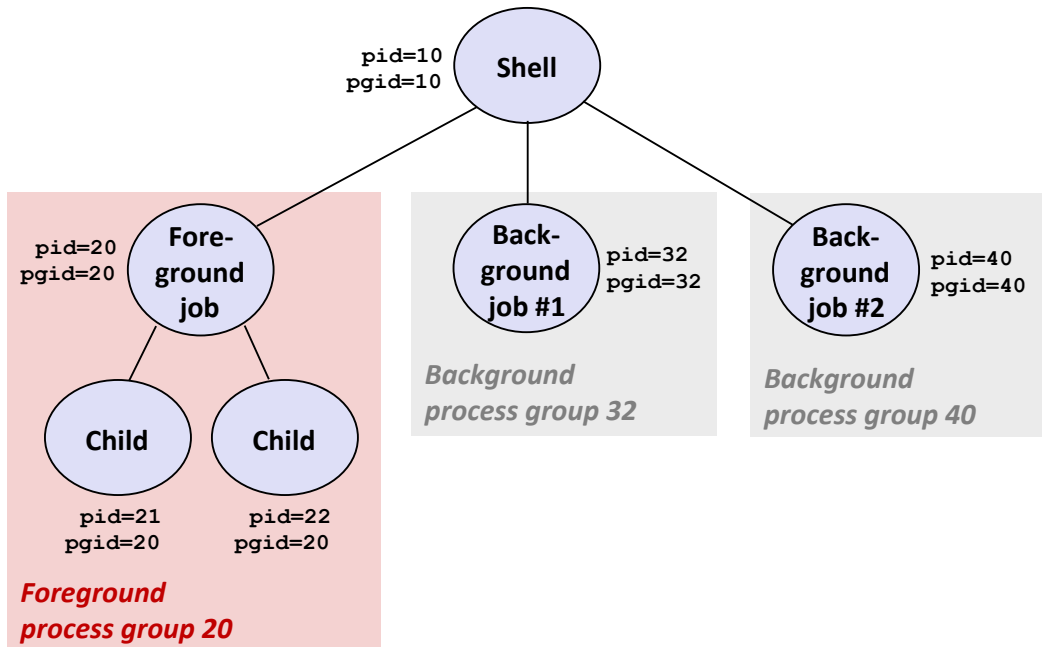
Signals

<i>ID</i>	<i>Name</i>	<i>Corresponding Event</i>	<i>Default Action</i>	<i>Can Override?</i>
2	SIGINT	Interrupt (Ctrl-C)	Terminate	Yes
9	SIGKILL	Kill process (immediately)	Terminate	No
11	SIGSEGV	Segmentation violation	Terminate & Dump	Yes
14	SIGALRM	Timer signal	Terminate	Yes
15	SIGTERM	Kill process (politely)	Terminate	Yes
17	SIGCHLD	Child stopped or terminated	Ignore	Yes
18	SIGCONT	Continue stopped process	Continue (Resume)	No
19	SIGSTOP	Stop process (immediately)	Stop (Suspend)	No
20	SIGTSTP	Stop process (politely)(Ctrl-Z)	Stop (Suspend)	Yes

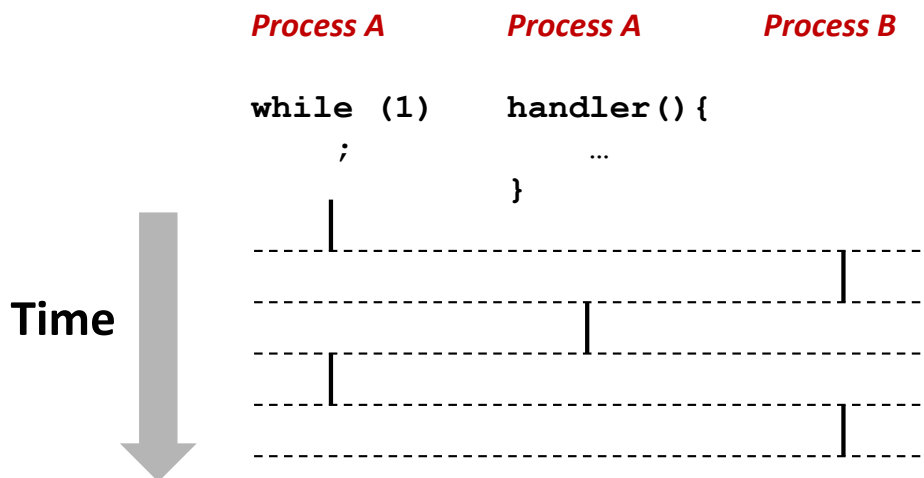
Signal Handler Control Flow



Process Groups



Signal Handler as Concurrent Flow



Concurrency Example (1)

```
int main(int argc, char** argv) {
    int pid;

    Signal(SIGCHLD, handler); // add handler
    initjobs(); /* Initialize the job list */

    while (1) {
        if ((pid = fork()) == 0) { /* Child */
            execve("/bin/date", argv, NULL);
        }
        addjob(pid); /* Add child to job list */
    }
    exit(0);
}
```

```
void handler(int sig) {
    pid_t pid;

    while ((pid = waitpid(-1, NULL, 0)) > 0) { /* Reap child */
        deletejob(pid); /* Delete the child from the job list */
    }
    if (errno != ECHILD) // handle case where waitpid returns
        unix_error("waitpid error"); // -1 but not an error
}
```

Concurrency Example (2)

```
int main(int argc, char** argv) {
    int pid;
    sigset_t mask_all, prev_all;
    sigfillset(&mask_all);
    Signal(SIGCHLD, handler);
    initjobs(); /* Initialize the job list */
    while (1) {
        if ((pid = fork()) == 0) { /* Child */
            execve("/bin/date", argv, NULL);
        }
        sigprocmask(SIG_BLOCK, &mask_all, &prev_all);
        addjob(pid); /* Add child to the job list */
        sigprocmask(SIG_SETMASK, &prev_all, NULL);
    }
    exit(0);
}
```

```
void handler(int sig) {
    sigset_t mask_all, prev_all;
    pid_t pid;
    sigfillset(&mask_all);
    while ((pid = waitpid(-1, NULL, 0)) > 0) { /* Reap child */
        sigprocmask(SIG_BLOCK, &mask_all, &prev_all);
        deletejob(pid); /* Delete child from job list */
        sigprocmask(SIG_SETMASK, &prev_all, NULL);
    }
    if (errno != ECHILD)
        unix_error("waitpid error");
}
```

Concurrency Example (3)

```
int main(int argc, char** argv) {
    int pid;
    sigset_t mask_all, mask_one, prev_one;

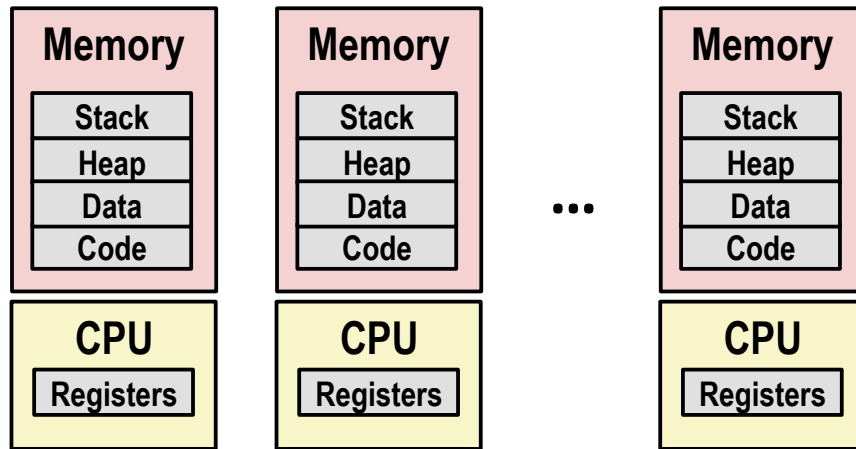
    sigfillset(&mask_all);
    sigemptyset(&mask_one);
    sigaddset(&mask_one, SIGCHLD);
    Signal(SIGCHLD, handler);
    initjobs(); /* Initialize the job list */

    while (1) {
        sigprocmask(SIG_BLOCK, &mask_one, &prev_one); /* Block SIGCHLD */
        if ((pid = fork()) == 0) { /* Child process */
            sigprocmask(SIG_SETMASK, &prev_one, NULL); /* Unblock SIGCHLD */
            execve("/bin/date", argv, NULL);
        }
        sigprocmask(SIG_BLOCK, &mask_all, NULL); /* Parent process */
        addjob(pid); /* Add the child to the job list */
        sigprocmask(SIG_SETMASK, &prev_one, NULL); /* Unblock SIGCHLD */
    }
    exit(0);
}
```

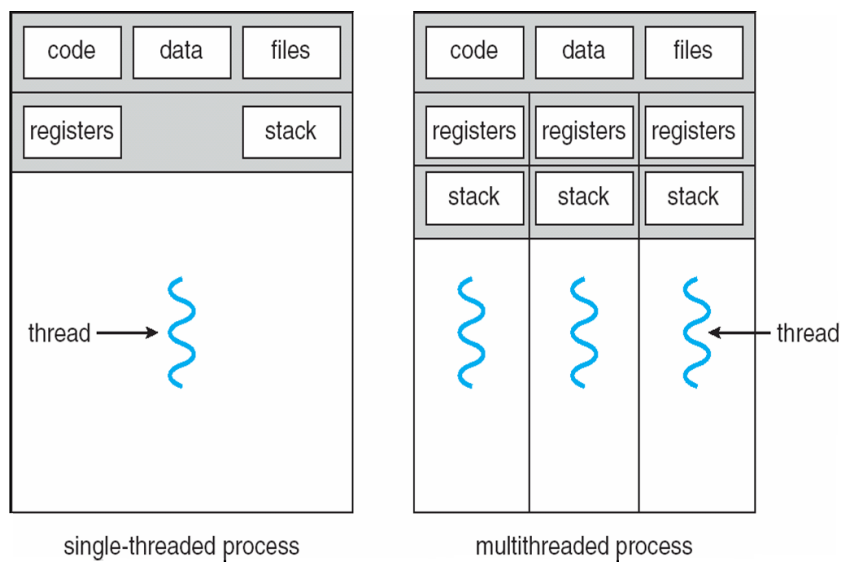
Key System Calls

- **fork** – Create a new process
- **execve** – Run a new program
- **kill** – Send a signal
- **waitpid** – Wait for and/or reap child process
- **setpgid** – Set process group ID
- **sigprocmask** – Block or unblock signals
 - **sigemptyset** – Create empty signal set
 - **sigfillset** – Add every signal number to set
 - **sigaddset** – Add signal number to set
 - **sigdelset** – Delete signal number from set
 - **sigsuspend** – Wait until signal received

Processes



Threads



single-threaded process

multithreaded process

Thread Example

