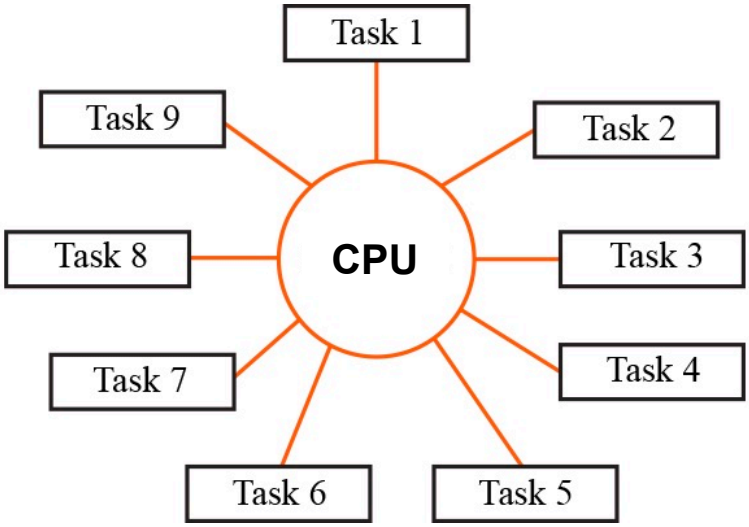
Control Flow



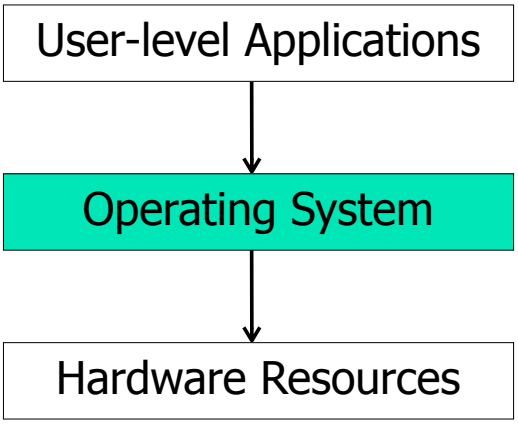
Physical Control Flow

Physical control flow

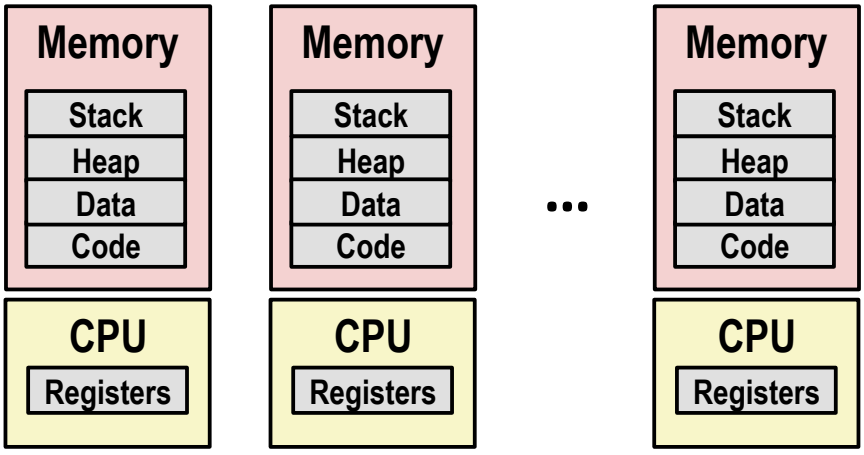
Time

<startup>
$inst_1$
$inst_2$
$inst_3$
…
$inst_n$
<shutdown>

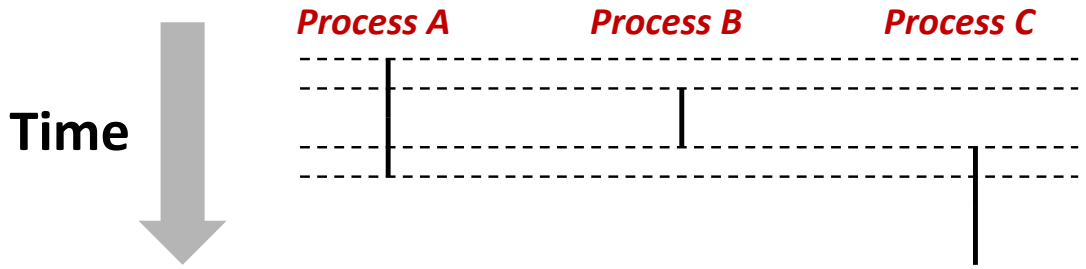
Operating System



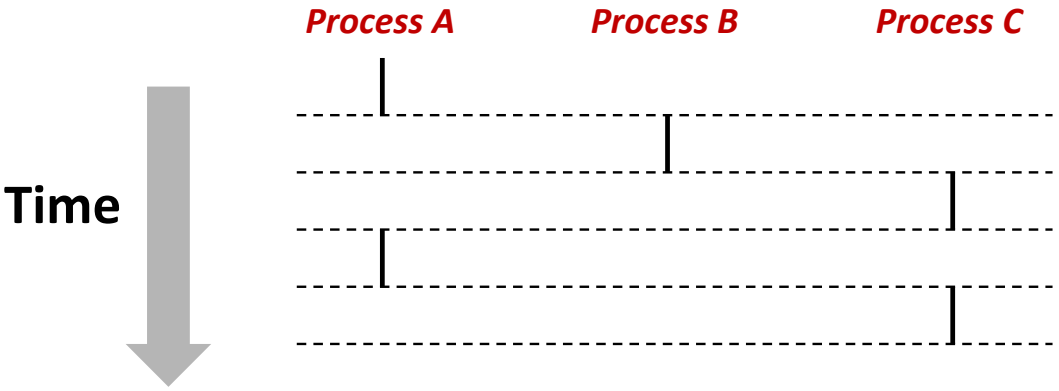
Processes



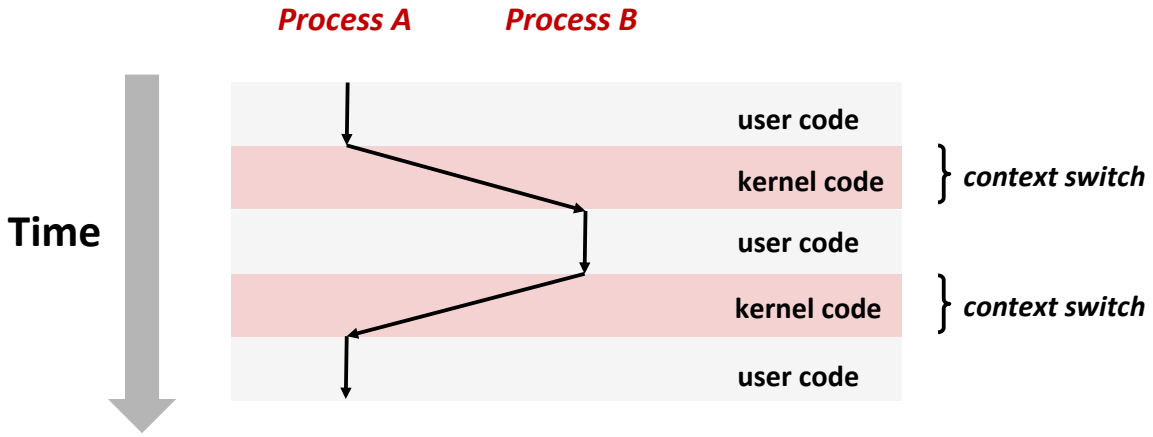
Control Flow Abstraction



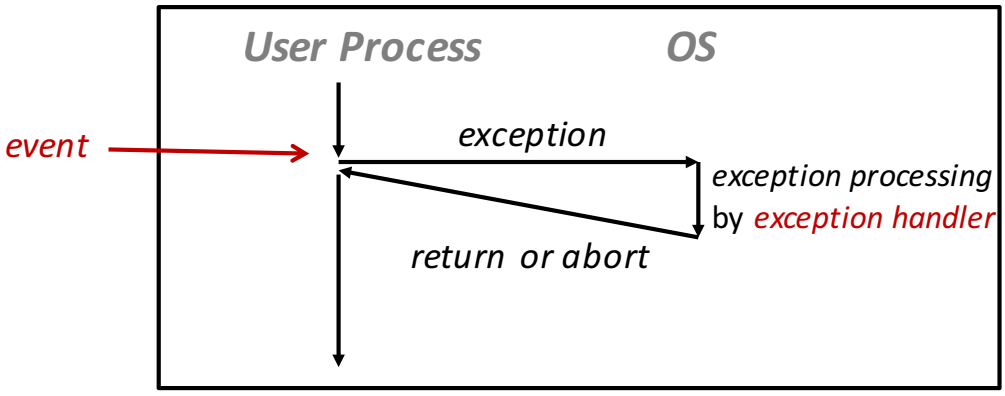
Control Flow Time-Sharing



Context Switching



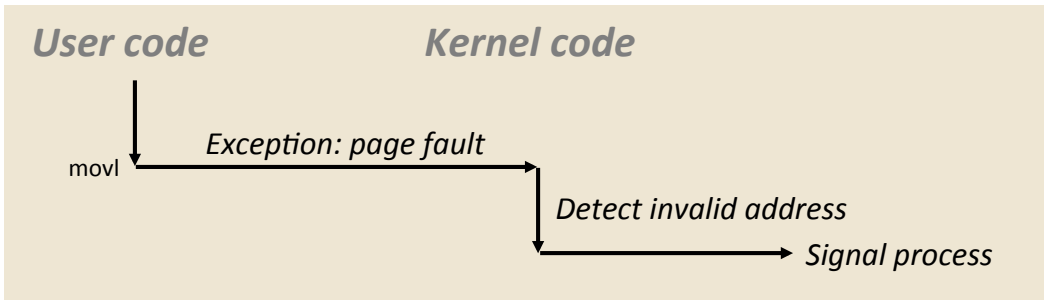
Exceptions



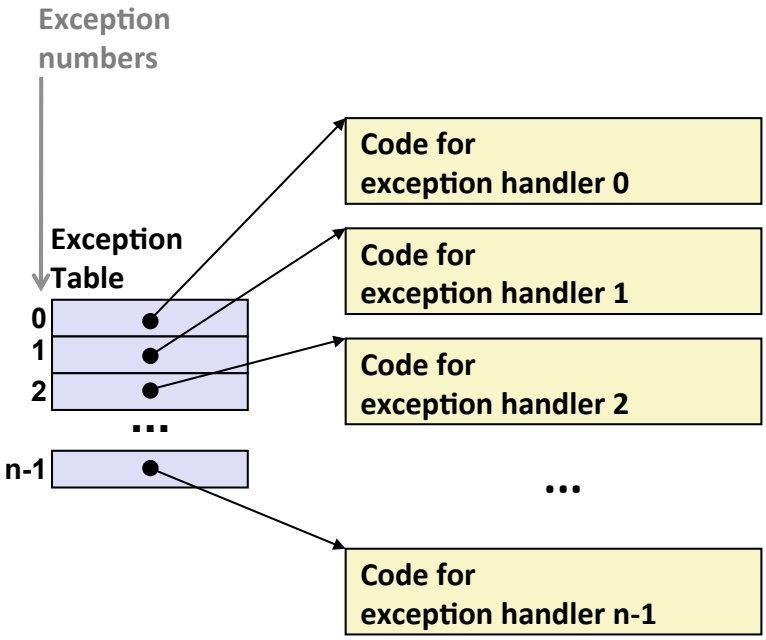
Example: Segmentation Fault

```
int a[1000];
main ()
{
    a[5000] = 13;
}
```

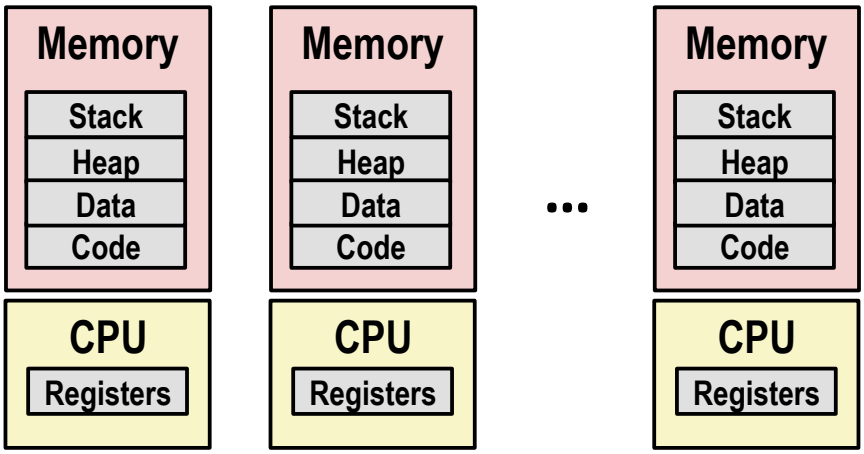
```
80483b7:      c7 05 60 e3 04 08 0d   movl   $0xd,0x804e360
```



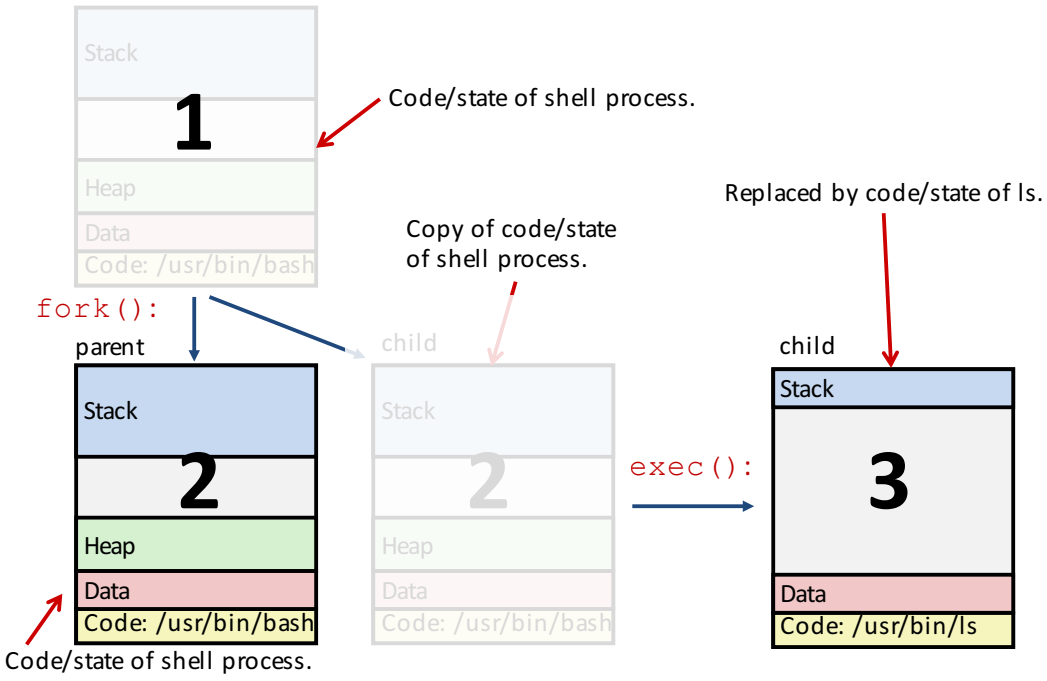
Exception Table



Process Management



Fork/Exec



Zombies!

Reaping: waitpid

```
pid_t waitpid(pid_t pid, int* stat, int ops)
```

wait set

Status Macros

```
pid_t waitpid(pid_t pid, int* stat, int ops)
```

WEXITSTATUS(stat) — child exit code

WIFEXITED(stat) — true if terminated normally (called exit or returned from main)

WIFSIGNALED(stat) — true if terminated by signal

WIFSTOPPED(stat) — true if paused by signal

Option Macros

```
pid_t waitpid(pid_t pid, int* stat, int ops)
```

WNOHANG — return immediately if child not already terminated

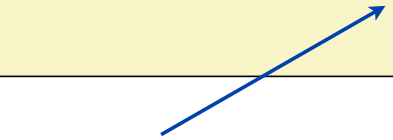
WUNTRACED — also wait for paused (stopped) children

WCONTINUED — also wait for resumed children

System Call Error Handling

Always check return values!

```
if ((pid = fork()) < 0) {
    fprintf(stderr, "fork error: %s\n", strerror(errno));
    exit(0);
}
```



Basic Shell Design

```
while (true) {
    Print command prompt.
    Read command line from user.
    Parse command line.
    If command is built-in, do it.
    Else fork process to execute command.
        in child:
            Execute requested command with execv.
                                    (never returns)
        in parent:
            Wait for child to complete.
}
```

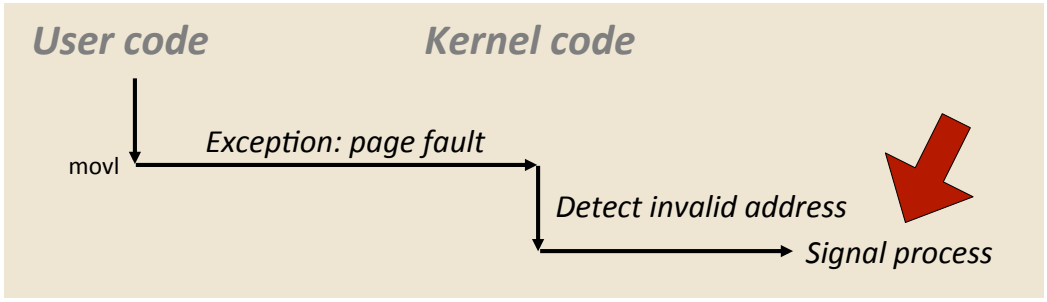
Signals

ID	Name	Corresponding Event	Default Action	Can Override?
2	SIGINT	Interrupt (Ctrl-C)	Terminate	Yes
9	SIGKILL	Kill process (immediately)	Terminate	**No**
11	SIGSEGV	Segmentation violation	Terminate	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

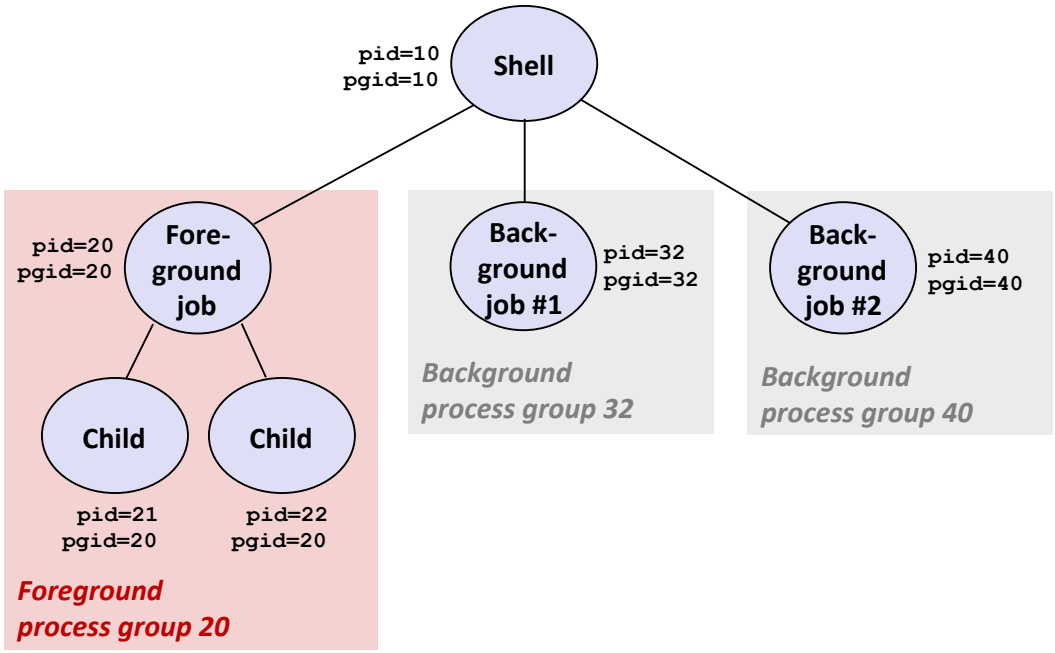
Recap: Segmentation Fault

```
int a[1000];
main ()
{
    a[5000] = 13;
}
```

```
80483b7:      c7 05 60 e3 04 08 0d   movl   $0xd,0x804e360
```

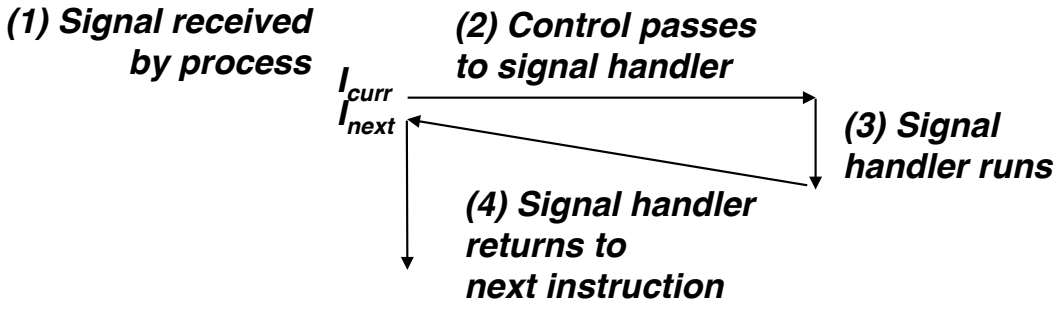


Process Groups

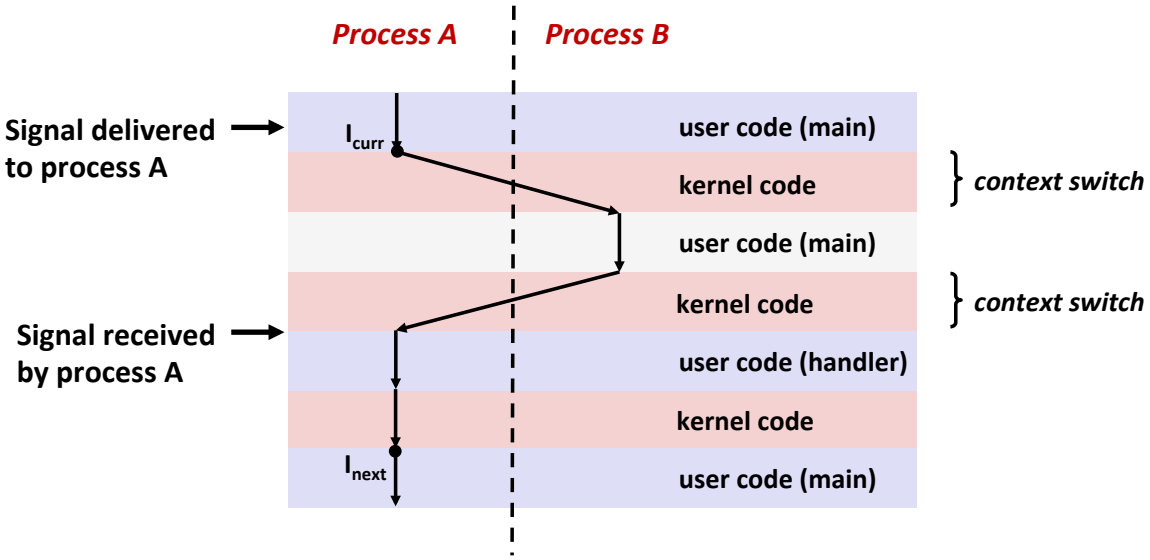


Signal Control Flow

(earlier: signal **delivered** to process)



Typical Signal Handler Control Flow



Reaping in Signal Handler

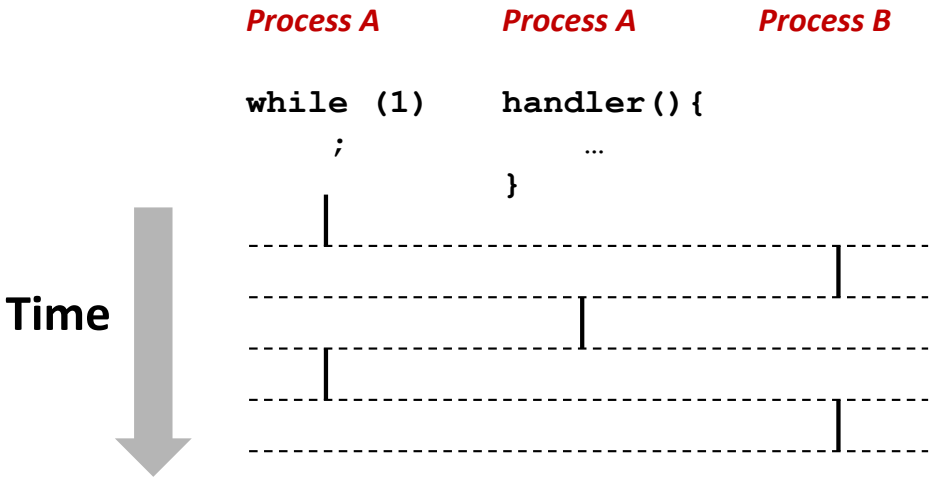
```
int main(int argc, char** argv) {
    int pid;

    Signal(SIGCHLD, sigchd_handler); // install signal handler

    while (1) {
        // print prompt, read cmd from user, etc.
        if ((pid = fork()) == 0) {
            execve(...); // child: run target program
        }
        // parent: wait for child to exit if foreground
    }
    return 0;
}
```

```
void sigchld_handler(int sig) {
    while ((pid = waitpid(-1, NULL, WNOHANG)) > 0) {
        // reaped child pid
    }
}
```

Signal Handler as Concurrent Flow



Job List Concurrency (1)

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

    Signal(SIGCHLD, sigchd_handler);
    initjobs(); // initialize job list

    while (1) {
        if ((pid = fork()) == 0) {
            execve(...);
        }
        addjob(pid); // add child to job list
    }
    return 0;
}
```

Concurrent job list modification!

```
void sigchld_handler(int sig) {
    while ((pid = waitpid(-1, NULL, WNOHANG)) > 0) {
        deletejob(pid); // delete child from job list
    }
}
```

Job List Concurrency (2)

```
int main(int argc, char** argv) {
    int pid;
    Signal(SIGCHLD, sigchd_handler);
    initjobs();
    sigset_t mask; // signal bit vector
    sigemptyset(&mask); // clear all bits
    sigaddset(&mask, SIGCHLD); // set SIGCHILD bit
    while (1) {
        if ((pid = fork()) == 0) {
            execve(...);
        }
        sigprocmask(SIG_BLOCK, &mask, NULL); // block SIGCHLD
        addjob(pid); // add child to job list
        sigprocmask(SIG_UNBLOCK, &mask, NULL); // unblock SIGCHLD
    }
    return 0;
}
```

Parent/child race condition!

```
void sigchld_handler(int sig) {
    while ((pid = waitpid(-1, NULL, WNOHANG)) > 0) {
        deletejob(pid); // delete child from job list
    }
}
```

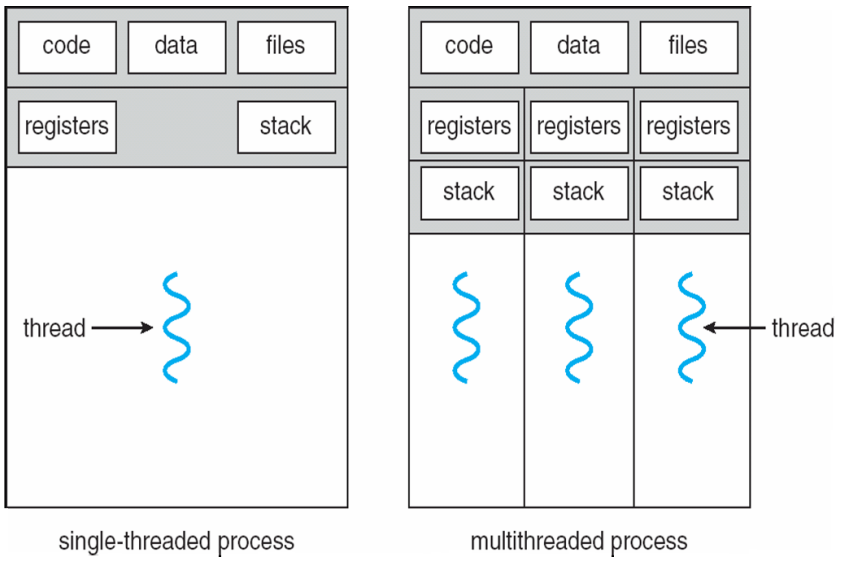

Job List Concurrency (3)

```
int main(int argc, char** argv) {
    int pid;
    Signal(SIGCHLD, sigchd_handler);
    initjobs();
    sigset_t mask; // signal bit vector
    sigemptyset(&mask); // clear all bits
    sigaddset(&mask, SIGCHLD); // set SIGCHILD bit
    while (1) {
        sigprocmask(SIG_BLOCK, &mask, NULL); // block SIGCHLD
        if ((pid = fork()) == 0) {
            // unblock in child (inherited from parent)
            sigprocmask(SIG_UNBLOCK, &mask, NULL);
            execve(...);
        }
        addjob(pid); // add child to job list
        sigprocmask(SIG_UNBLOCK, &mask, NULL); // unblock SIGCHLD
    }
    return 0;
}
```

Useful 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
- **`sigsuspend`** – Wait until signal received
- **`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

Threads



Thread Example

```c
/*
 * hello.c - Pthreads "hello, world" program
 */

void* thread(void* vargp);

int main() {

    pthread_t tid;
    pthread_create(&tid, NULL, thread, NULL);
    pthread_join(tid, NULL);
    exit(0);
}
```

hello.c

Thread ID

Thread attributes (usually NULL)

Thread routine

*Thread arguments (void *p)*

*Return value (void **p)*

```c
void* thread(void* vargp) { /* thread routine */
    printf("Hello, world!\n");
    return NULL;
}
```

hello.c