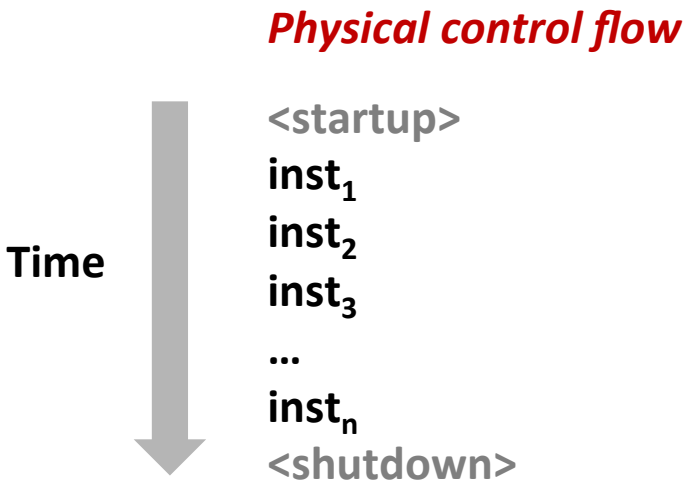
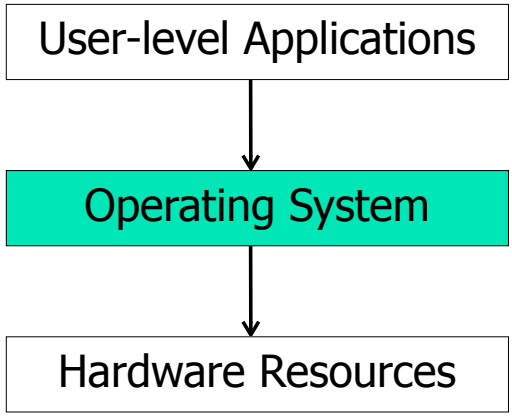
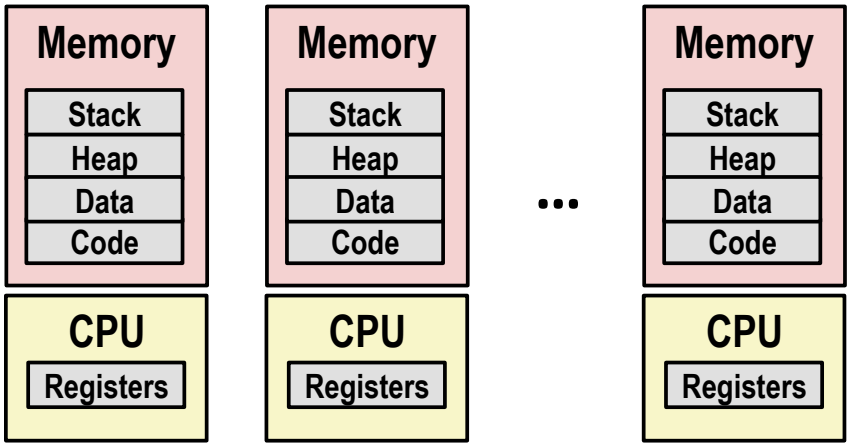
Physical Control Flow



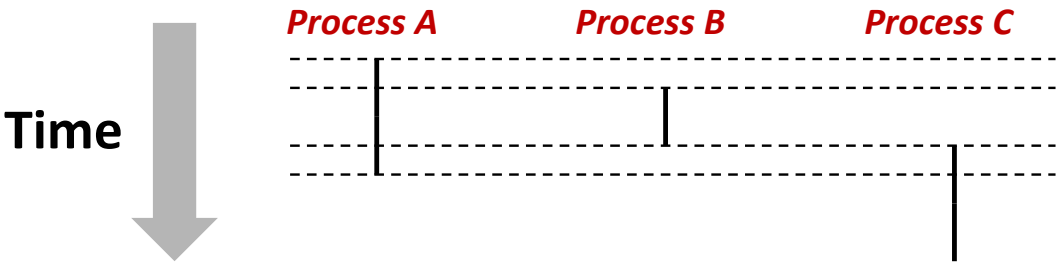
Operating System



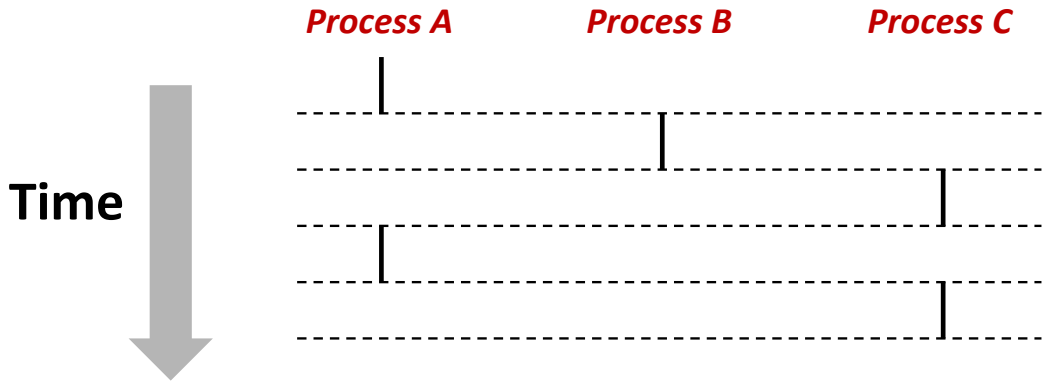
Processes



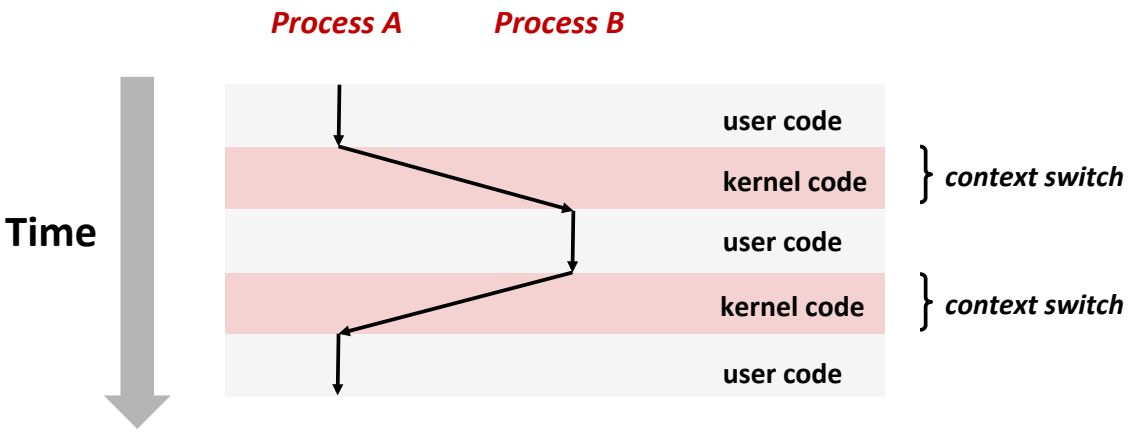
Control Flow Abstraction



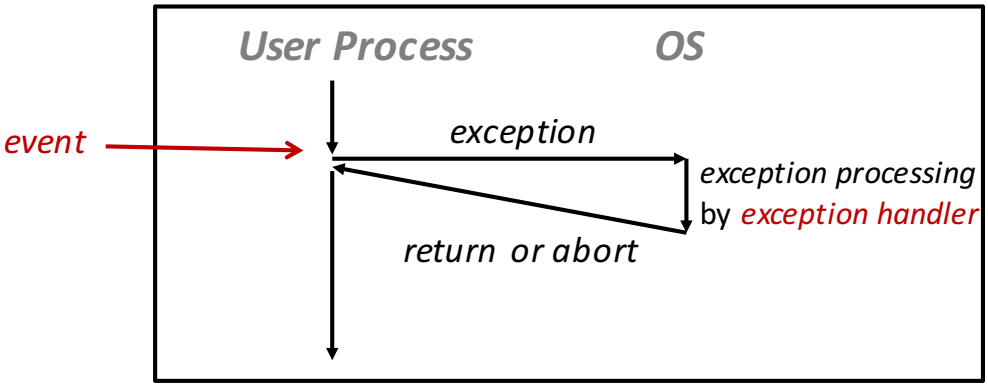
Control Flow Time-Sharing



Context Switching



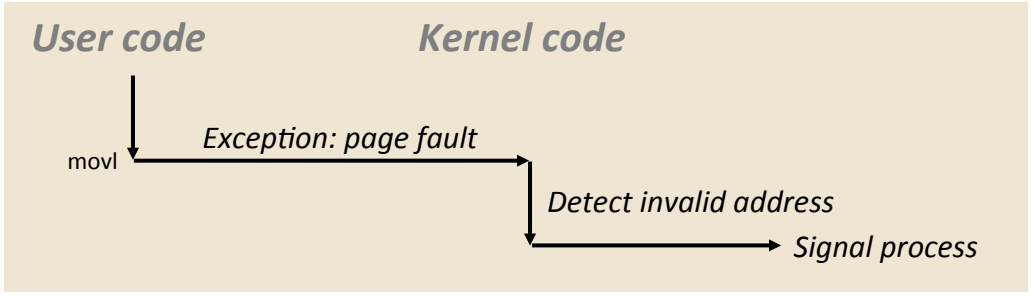
Exceptions



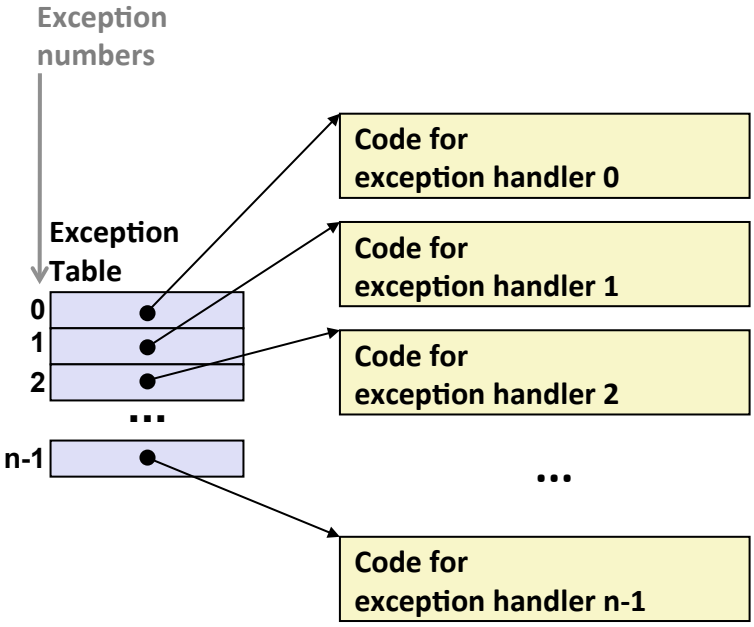
Example: Segmentation Fault

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

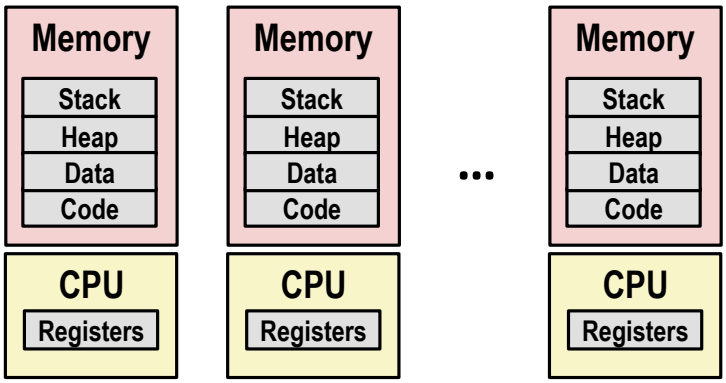
```
80483b7:      c7 05 60 e3 04 08 0d   movl   $0xd,0x804e360
```



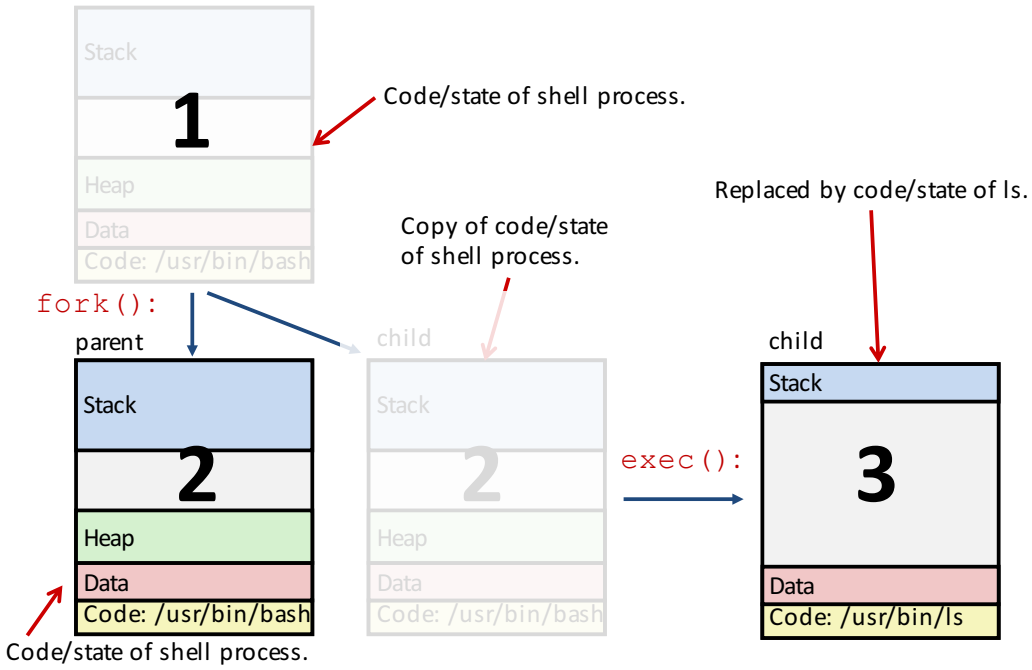
Exception Table



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

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

WIFSIGNALED(stat) — true if terminated by signal

true if paused by signal — **WIFSTOPPED(stat)**

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.
}
```

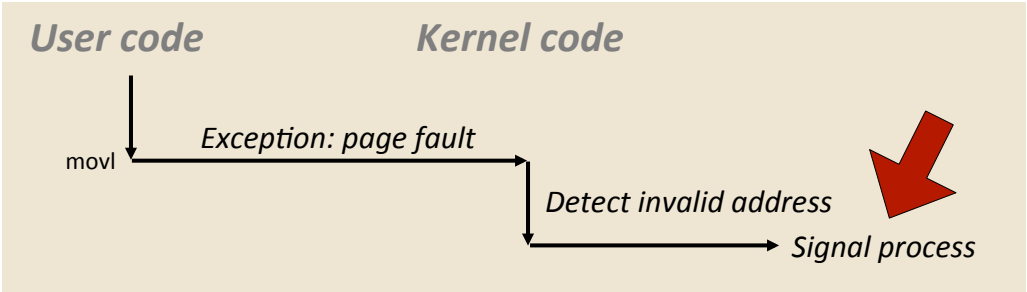

Signals

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

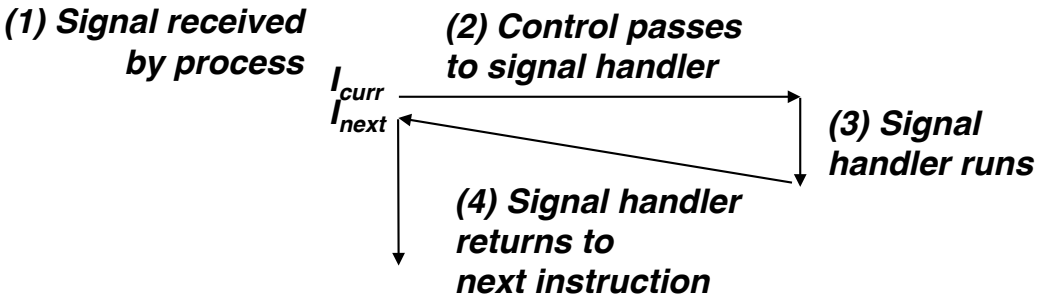
Recap: Segmentation Fault

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

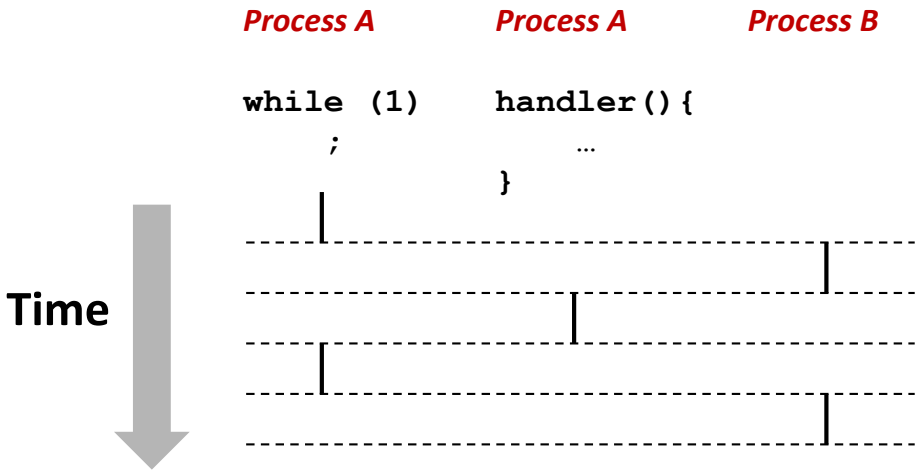
```
80483b7:      c7 05 60 e3 04 08 0d   movl   $0xd,0x804e360
```



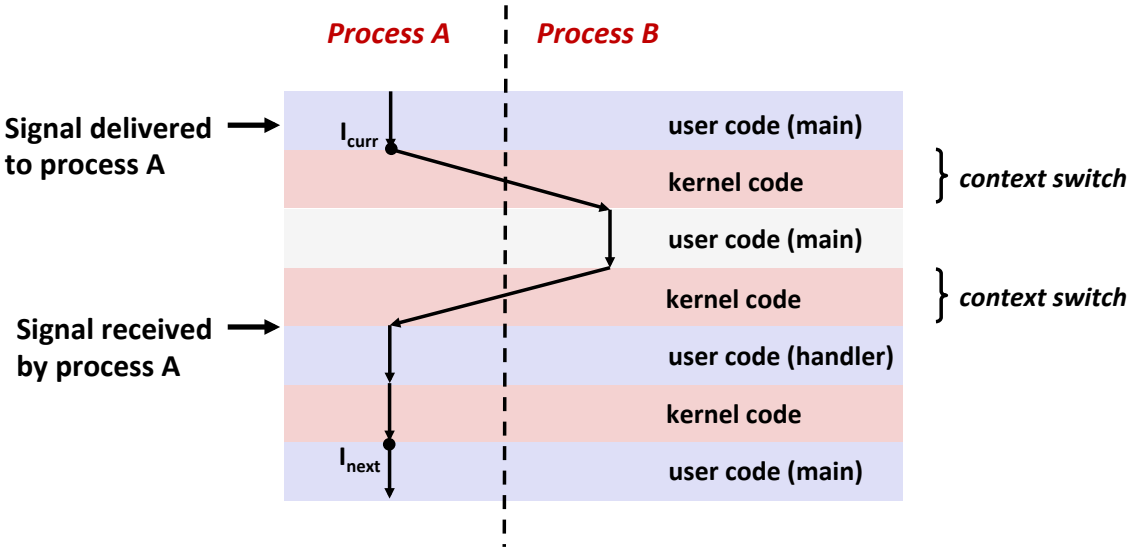
Signal Control Flow



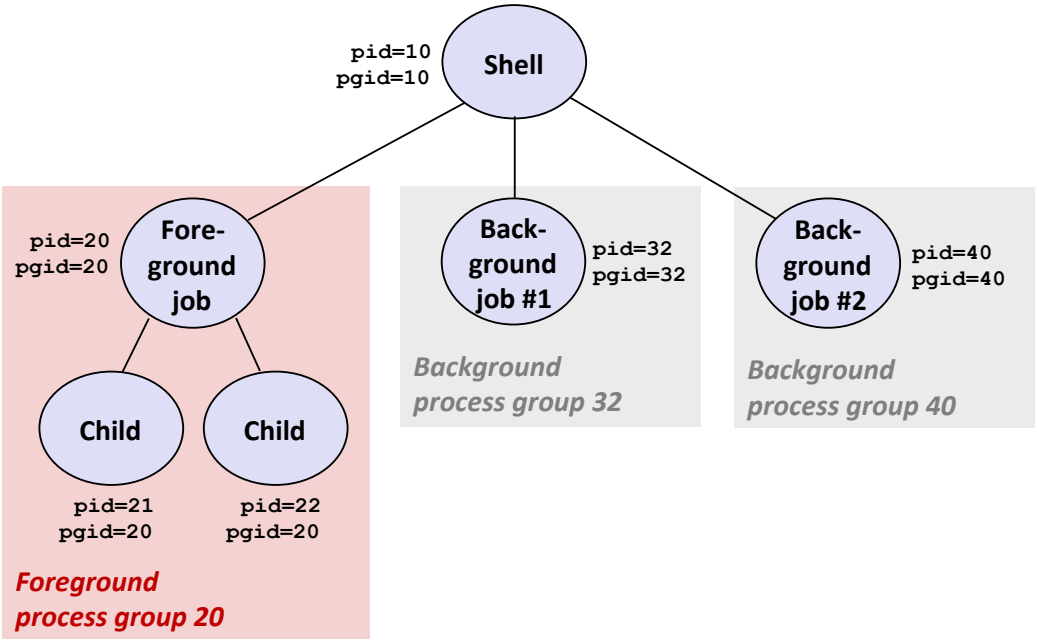
Signal Handler as Concurrent Flow



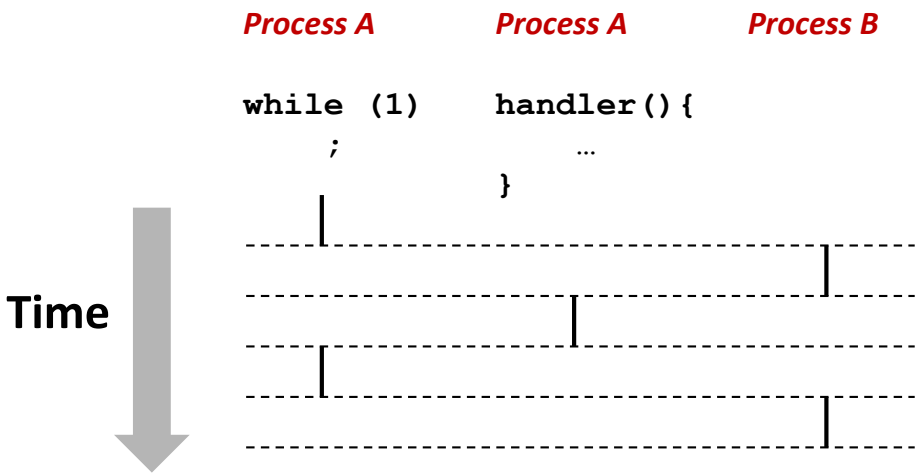
Signal Handler as Concurrent Flow (alt)



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);
}
```

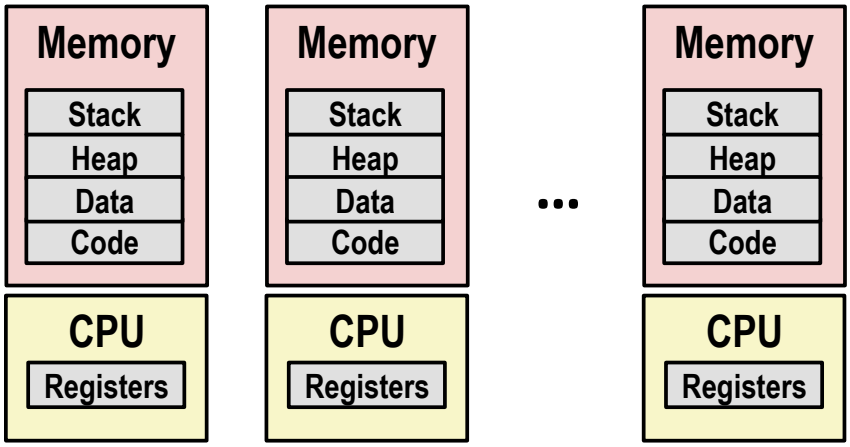

Key System Calls

- **`fork`** – Create a new process
- **`execve`** – Run a new program

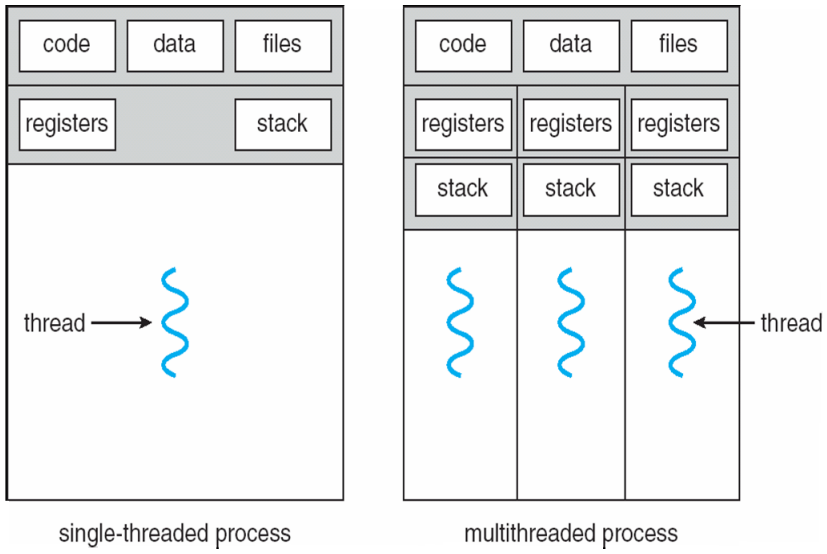
- **`kill`** – Send a signal
- **`waitpid`** – Wait for and/or reap child process
- **`setpgid`** – Set process group ID

- **`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



Thread Example

```
/*
 * 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)*

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

hello.c