

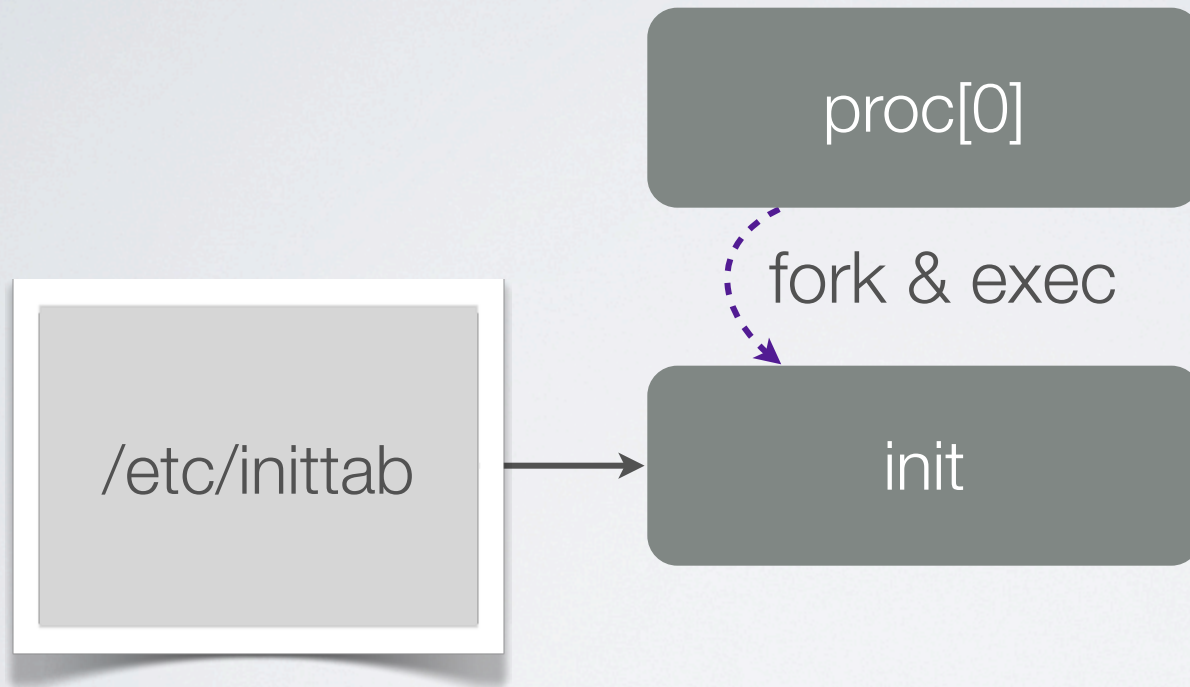
Exceptional Control Flow

CS351 : Saelee

UNIX family tree

proc[0]

“handcrafted”

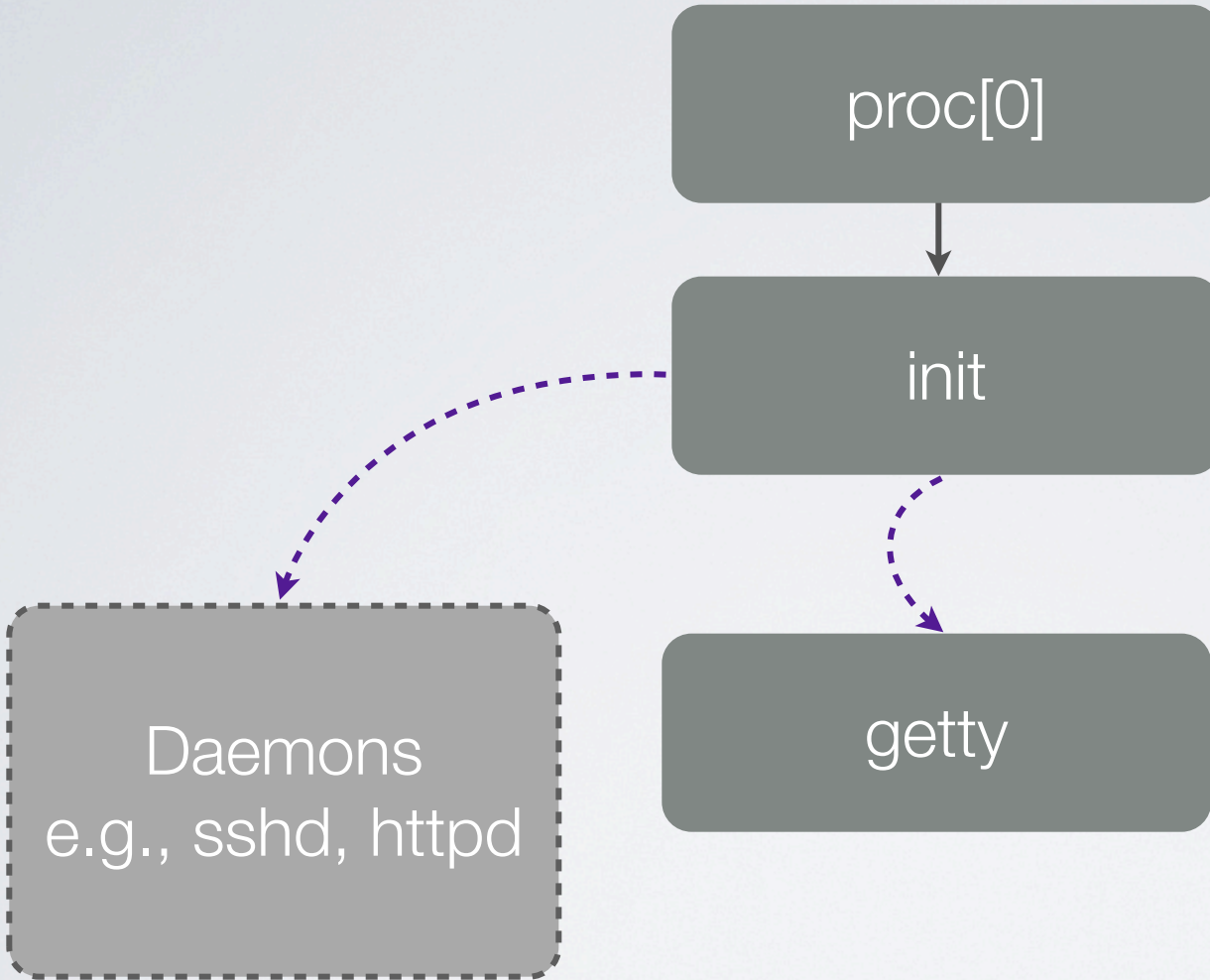


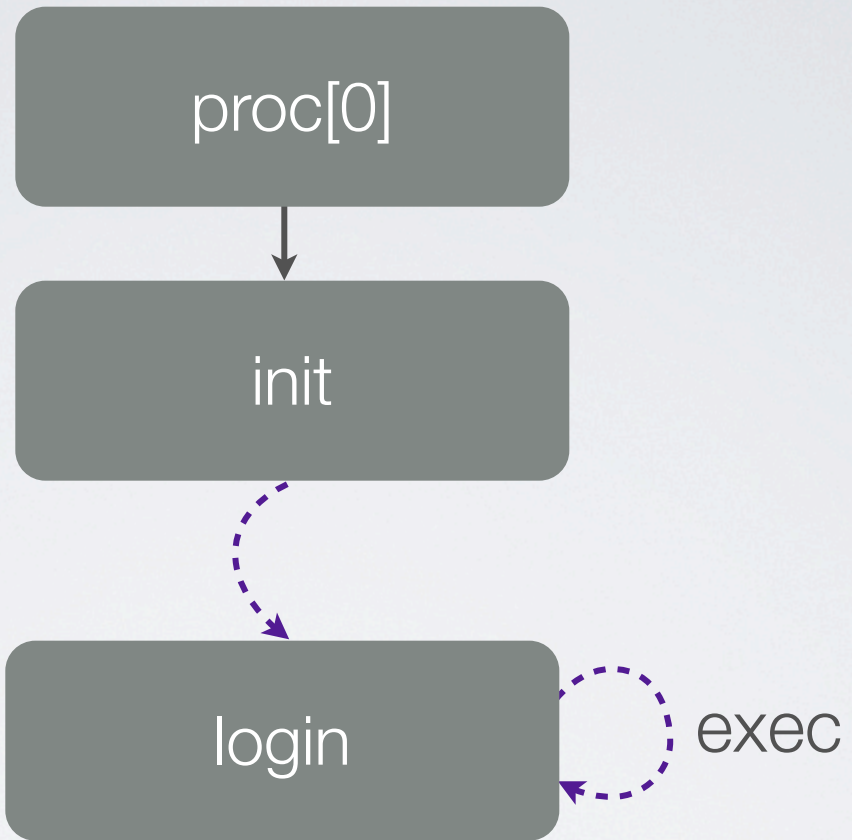
proc[0]

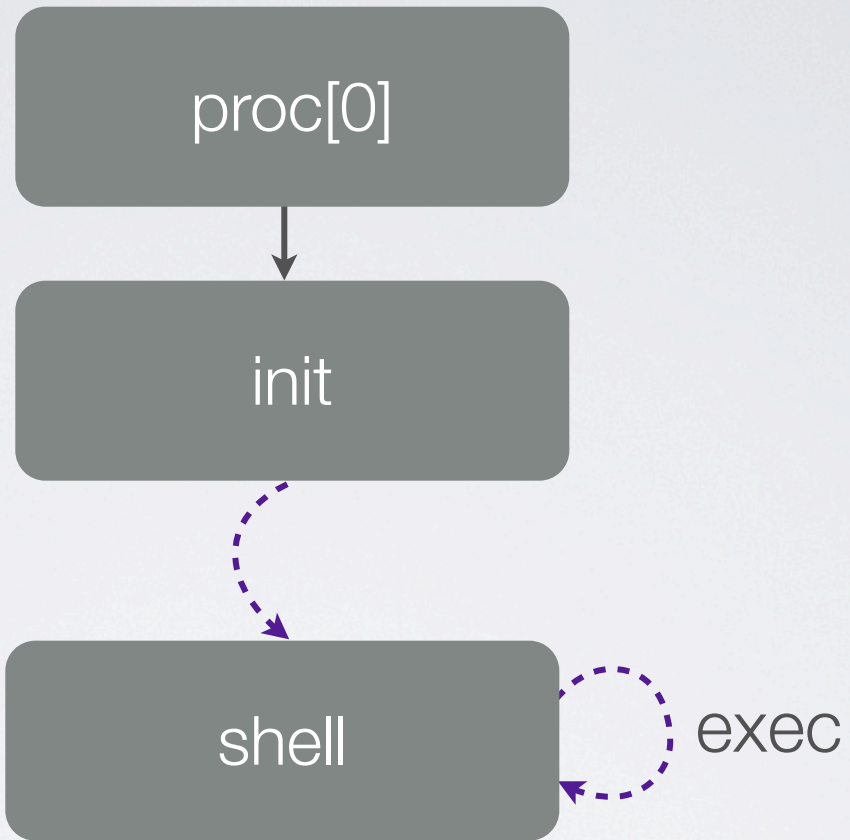
init

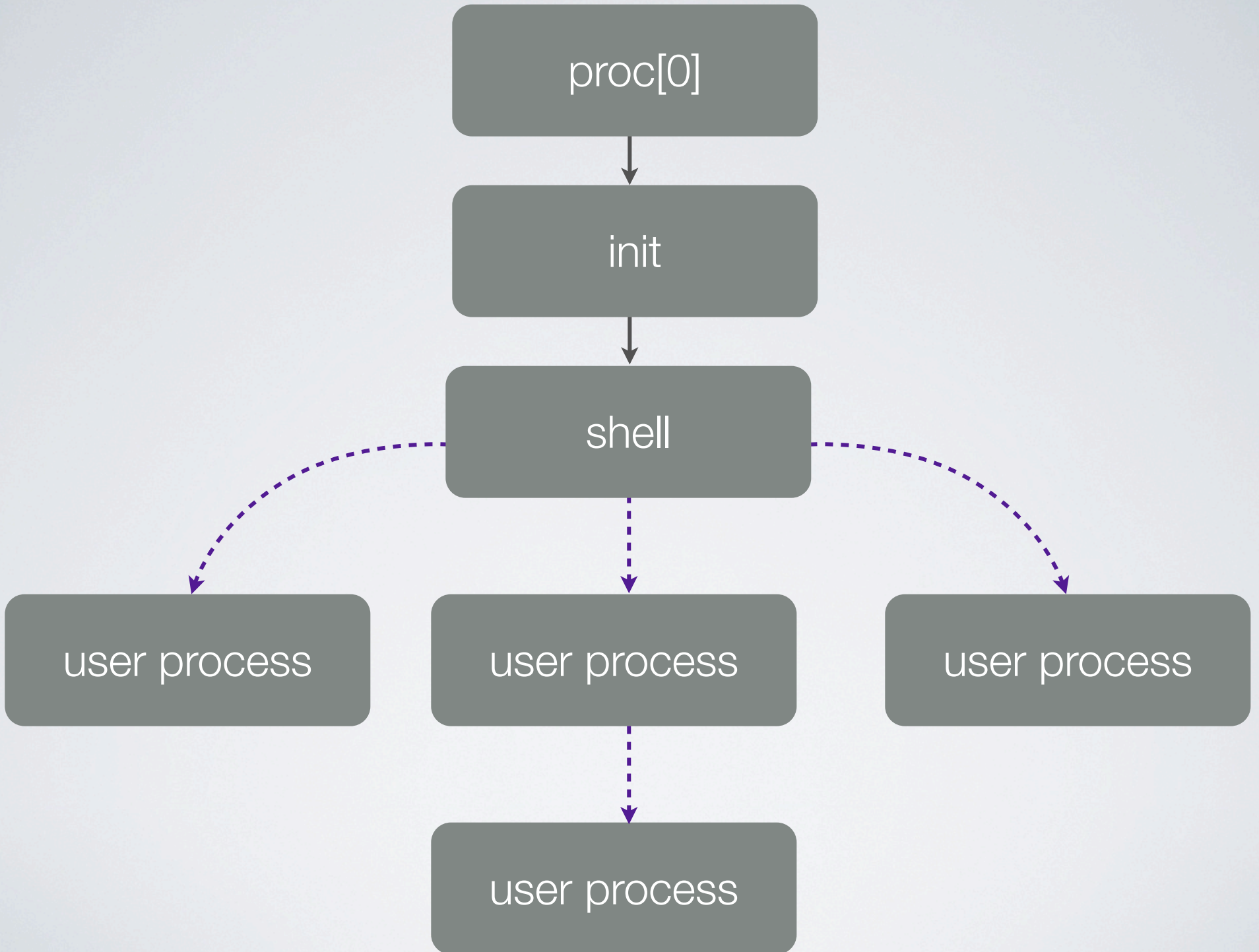
getty

Daemons
e.g., sshd, httpd









shell



read-eval loop

```
/* read-eval loop */
while (!feof(stdin)) {
    printf(">"); /* print prompt */

    /* read command and remove newline */
    fgets(buf, 100, stdin);
    for (i=strlen(buf)-1; buf[i]!='\n'; buf[i--]=0);

    if (strcmp(buf, "quit") == 0) {
        break;
    } else {
        /* fork and run command in child */
        if ((pid = fork()) == 0) {
            if (execlp(buf, buf, (char *)0) < 0) {
                printf("Command not found\n");
                exit(0);
            }
        }

        /* wait for completion in parent */
        waitpid(pid, NULL, 0);
    }
}
```

Demo



```
./simple_shell1  
ls  
ps  
quit
```

background jobs

long-running, non-interactive
GUI-based

job management

shell convention:

> prog &

```
/* read command and remove newline */
fgets(buf, 100, stdin);
for (i=strlen(buf)-1; buf[i]=='\n'; buf[i--]=0);

bg = (buf[i] == '&');
if (bg)
    buf[i] = 0;

if (strcmp(buf, "quit") == 0) {
    break;
} else {
    /* fork and run command in child */
    if ((pid = fork()) == 0) {
        if (execlp(buf, buf, (char *)0) < 0) {
            printf("Command not found\n");
            exit(0);
        }
    }
}

if (!bg) {
    /* wait for completion in parent */
    waitpid(pid, NULL, 0);
}
```

Demo



```
./simple_shell2  
ls  
ls&  
ps  
quit
```



background zombie

```
while (wait(NULL) > 0);
```

asynchronous notification

software “interrupt”

signals

system events

ctrl-C, ctrl-Z,
invalid memory access

explicitly sent

```
#include <signal.h>
```

```
int kill (pid_t pid, int sig);
```

No	Name	Default Action	Description
1	SIGHUP	terminate process	terminal line hangup
2	SIGINT	terminate process	interrupt program
3	SIGQUIT	create core image	quit program
4	SIGILL	create core image	illegal instruction
5	SIGTRAP	create core image	trace trap
6	SIGABRT	create core image	abort program (formerly SIGIOT)
7	SIGEMT	create core image	emulate instruction executed
8	SIGFPE	create core image	floating-point exception
9	SIGKILL	terminate process	kill program
10	SIGBUS	create core image	bus error
11	SIGSEGV	create core image	segmentation violation
12	SIGSYS	create core image	non-existent system call invoked
13	SIGPIPE	terminate process	write on a pipe with no reader
14	SIGALRM	terminate process	real-time timer expired
15	SIGTERM	terminate process	software termination signal
16	SIGURG	discard signal	urgent condition present on socket
17	SIGSTOP	stop process	stop (cannot be caught or ignored)
18	SIGTSTP	stop process	stop signal generated from keyboard
19	SIGCONT	discard signal	continue after stop
20	SIGCHLD	discard signal	child status has changed



```
/* non-terminating children */
for (i = 0; i < 5; i++)
    if ((pid[i] = fork()) == 0)
        while(1);

/* kill children with SIGINT */
for (i = 0; i < 5; i++)
    kill(pid[i], SIGINT);

/* reap children */
for (i = 0; i < 5; i++) {
    wpid = wait(&status);
    if (WIFSIGNALED(status))
        printf("Child %d terminated with sig %d\n",
            wpid, WTERMSIG(status));
}
```

```
Child 3606 terminated with sig 2
Child 3609 terminated with sig 2
Child 3608 terminated with sig 2
Child 3607 terminated with sig 2
Child 3605 terminated with sig 2
```

```
> ./proc &
[1] 3057
> ps
  PID STAT  TT  STAT      TIME COMMAND
 1520 S      s000 S      0:01.63 -zsh
 3057 S      s000 S      0:00.00 ./proc
> kill -s SIGINT 3057
[1] + 3057 interrupt ./proc
>
```

multiple recipients

process groups

shell

pid=10, pgid=10

user process

pid=11, pgid=10



shell

pid=10, pgid=10

user process

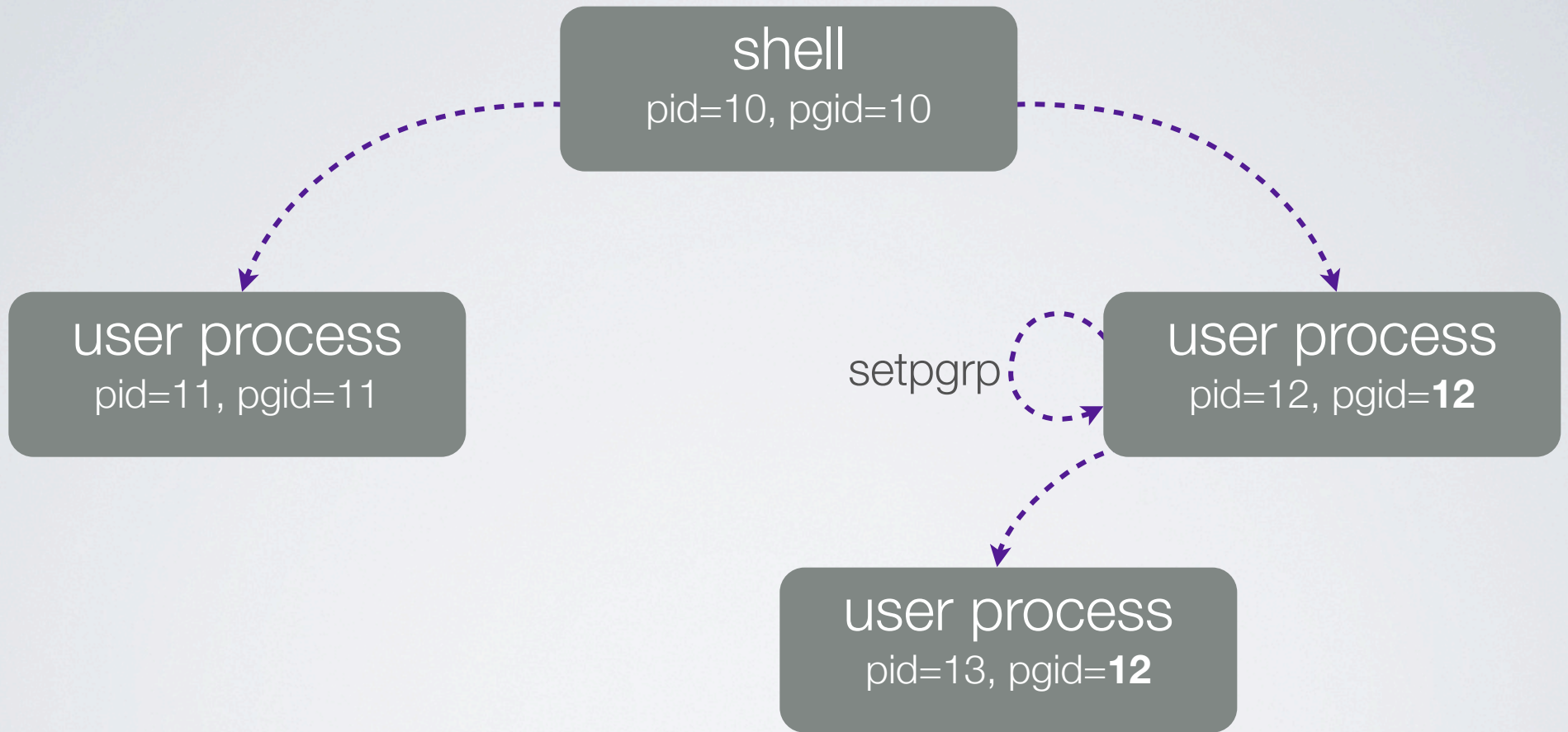
pid=11, pgid=**11**

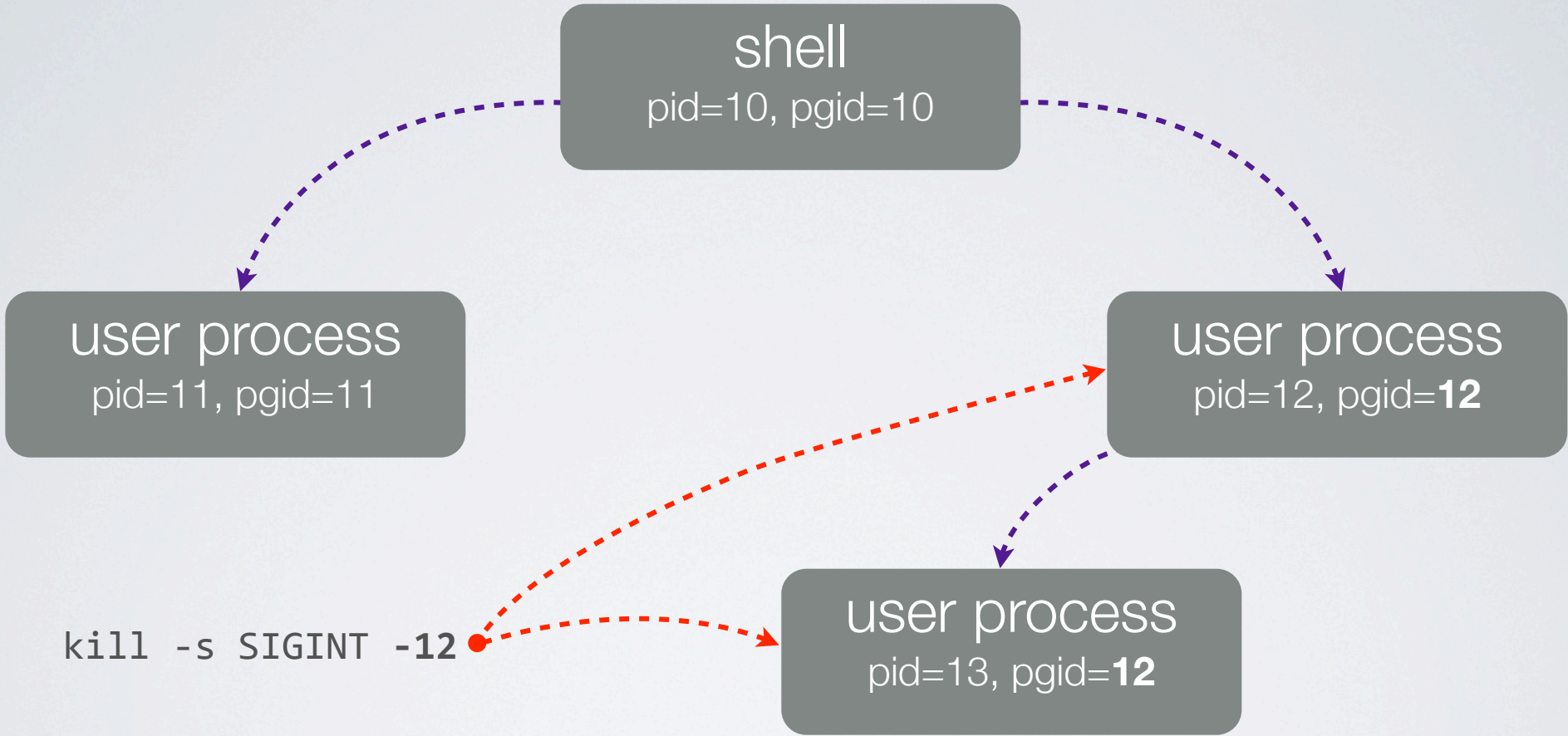
setpgrp

user process

pid=12, pgid=10







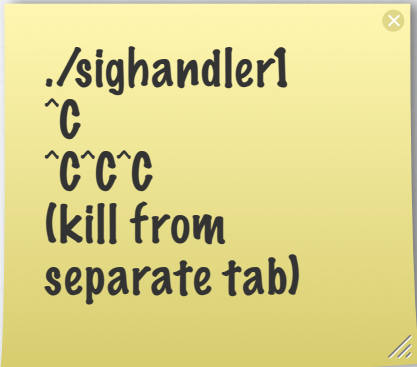
signal handlers

```
typedef void (*sig_t) (int);  
sig_t signal(int sig, sig_t func);
```

`sig_t` = “callback” function

```
void sigint_handler (int sig) {  
    printf("Signal %d received\n", sig);  
    sleep(1);  
}  
  
int main () {  
    signal(SIGINT, sigint_handler);  
    while (1)  
        pause();  
}
```

Demo



```
./sighandler1  
^C  
^C^C  
(kill from  
separate tab)
```

kernel accounting

“pending” bit vector

“blocked” bit vector

`sigprocmask`

pending & ~blocked

signals aren't queued!

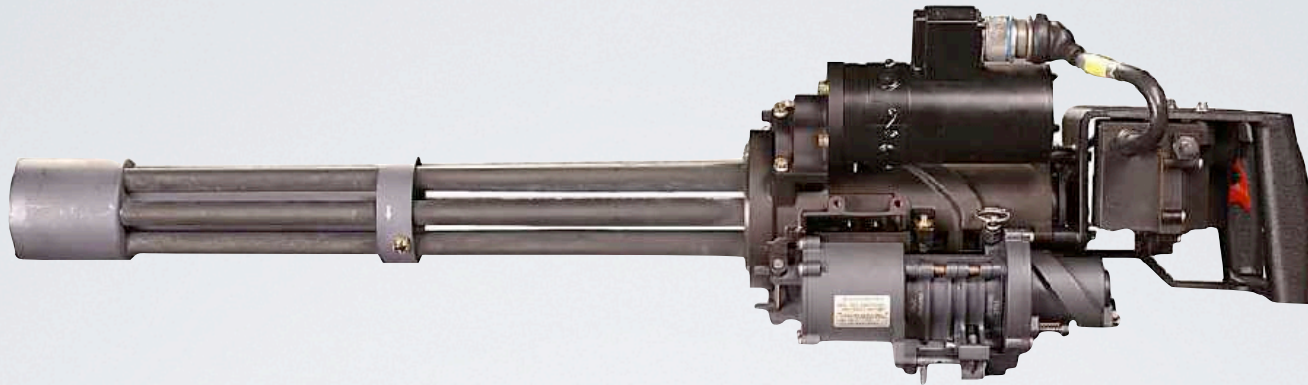


background zombie

```
void sigchld_handler (int sig) {  
    int pid, status;  
    if ((pid = wait(&status)) > 0) {  
        printf("Process %d reaped!\n", pid);  
    }  
}
```



background zombies?



```
while ((pid = waitpid(-1, &status, WNOHANG|WUNTRACED)) > 0)
```

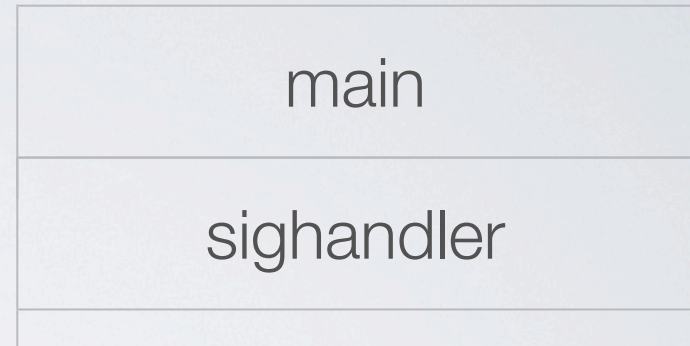
non-blocking

Stack

```
int count = 0;

void sighandler (int sig) {
    printf("tick...\n");
    count++;
}

int main () {
    signal(SIGALRM, sighandler);
    while (count < 3) {
        kill(getpid(), SIGALRM);
        sleep(1);
    }
    printf("BOOM!\n");
    return 0;
}
```



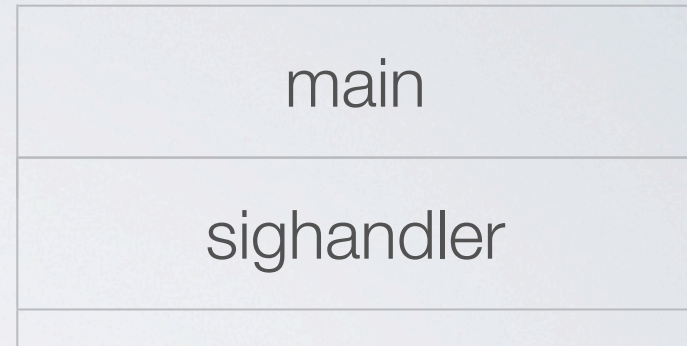
tick...

Stack

```
int count = 0;

void sighandler (int sig) {
    printf("tick...\n");
    count++;
}

int main () {
    signal(SIGALRM, sighandler);
    while (count < 3) {
        kill(getpid(), SIGALRM);
        sleep(1);
    }
    printf("BOOM!\n");
    return 0;
}
```



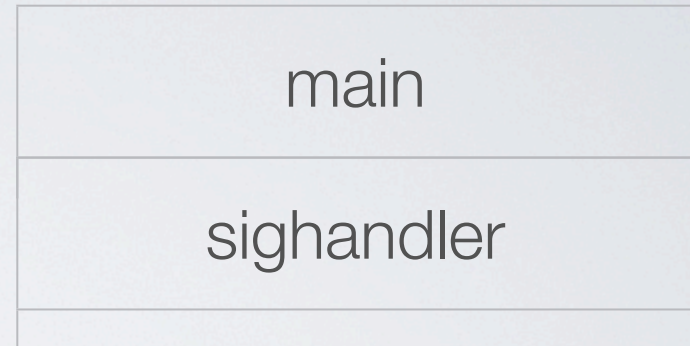
```
tick...
tick...
```

Stack

```
int count = 0;

void sighandler (int sig) {
    printf("tick...\n");
    count++;
}

int main () {
    signal(SIGALRM, sighandler);
    while (count < 3) {
        kill(getpid(), SIGALRM);
        sleep(1);
    }
    printf("BOOM!\n");
    return 0;
}
```



```
tick...
tick...
tick...
```

Stack

```
int count = 0;

void sighandler (int sig) {
    printf("tick...\n");
    count++;
}

int main () {
    signal(SIGALRM, sighandler);
    while (count < 3) {
        kill(getpid(), SIGALRM);
        sleep(1);
    }
    printf("BOOM!\n");
    return 0;
}
```



```
tick...
tick...
tick...
BOOM!
```

Exceptional Control Flow

within stack discipline

overhead?

trap
+ signal delivery
+ context switch(es)

expensive!

Bonus: Intra-process ECF

non-local jump

environment buffer

```
int setjmp (jmp_buf env);  
save environment  
(return 0)
```

```
void longjmp (jmp_buf env, int val);  
return to saved environment  
(return val)
```

```
jmp_buf buf;
```

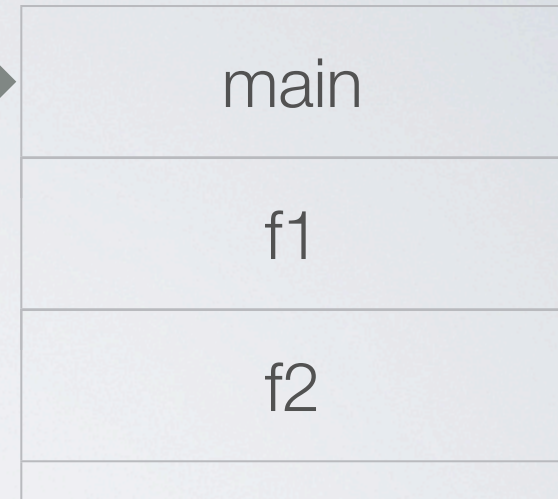
```
void f1 (int x) {  
    printf("In f1\n");  
    f2(x);  
}
```

```
void f2 (int x) {  
    printf("In f2\n");  
    if (!x)  
        longjmp(buf, 1);  
}
```

```
int main () {  
    int i = 0;  
    if (setjmp(buf) != 0) {  
        printf("Restarting...\n");  
    }  
    f1(i++);  
    printf("Exiting.\n");  
    return 0;  
}
```



Stack



```
In f1  
In f2  
Restarting
```

```
jmp_buf buf;
```

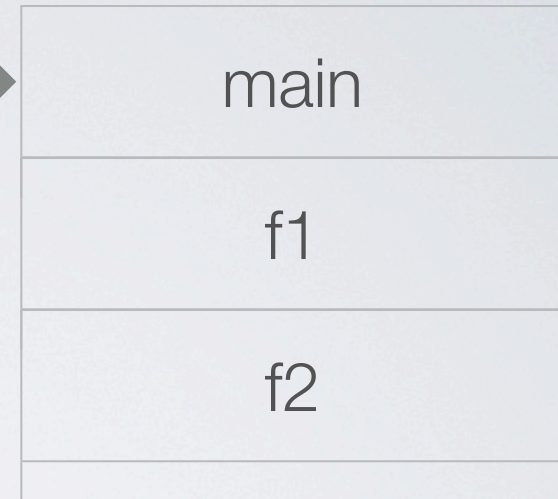
```
void f1 (int x) {  
    printf("In f1\n");  
    f2(x);  
}
```

```
void f2 (int x) {  
    printf("In f2\n");  
    if (!x)  
        longjmp(buf, 1);  
}
```

```
int main () {  
    int i = 0;  
    if (setjmp(buf) != 0) {  
        printf("Restarting...\n");  
    }  
    f1(i++);  
    printf("Exiting.\n");  
    return 0;  
}
```



Stack



```
In f1  
In f2  
Restarting  
In f1  
In f2  
Exiting
```

limited by stack

backtrace
exception handling

linked-list of buffers