Process Termination

When does a process die?

- **A process terminates for one of 3 reasons:**
  - It calls exit();
  - It returns (an int) from main
  - It receives a signal (from the OS or another process) whose default action is to terminate

- **Key observation: the dying process *produces status information*.**
  - Who looks at this? The parent process!

- **`void exit(int status);`**
  - Terminates a process with a specified status
  - By convention, status of 0 is normal exit, non-zero indicates an error of some kind

```
void foo() {
  exit(1); /* no return */
}

int main() {
  foo();   /* no return */
  return 0;
}
```

Terminal

why_wait.c (~/Desktop/SS 2021/Wk-4/example code/wait) - gedit

Open Save Undo

why_wait.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

main()
{ int pid;

  printf("I'm the original process with PID %d and PPID %d. \n", getpid(), getppid() );
  pid = fork(); /* Duplicate. Child and parent continue from here */

  if ( pid!= 0 ) /* pid is non-zero, so the parent - Parent-Child execute from this point */  {
      printf("I'm the parent process with PID %d and PPID %d. \n", getpid(), getppid());
      printf("My child's PID is %d \n", pid );
  }
  else /* pid is zero, so I must be the child */
  {
  printf("I'm the child process with PID %d and PPID %d. \n", getpid(), getppid());
  }
  printf("PID %d terminates. \n", getpid()); /* Both processes execute this */
}
```

jayprakash@jayprakash-System-AsusPG500: ~/Desktop/SS 2021/Wk-4/example code/wait

```
$ gcc why_wait.c -o why_wait
$ ./why_wait
I'm the original process with PID 3690 and PPID 3535.
I'm the parent process with PID 3690 and PPID 3535.
My child's PID is 3691
PID 3690 terminates.
I'm the child process with PID 3691 and PPID 3690.
PID 3691 terminates.
$
```

It is dangerous for a parent process to terminate without waiting for the death of its child process. The only reason our program doesn't wait for its child to terminate is because we haven't yet used the "wait()" system call!.

What should happen if dead child processes are never reaped? (That is, the parent has not waited() on them.)

1. **The OS should remove them from the process table**
2. **The OS should leave them in the process table**
3. **Do nothing**

Zombies

- **Zombie: A process that has terminated but not been reaped by its parent (AKA defunct process)**

- **“dead” but still tracked by the OS**
  - Parent may still reap them, want to know status
  - Don’t want to re-use the process ID yet

- **Does not respond to signals (can’t be killed)**

Reaping children

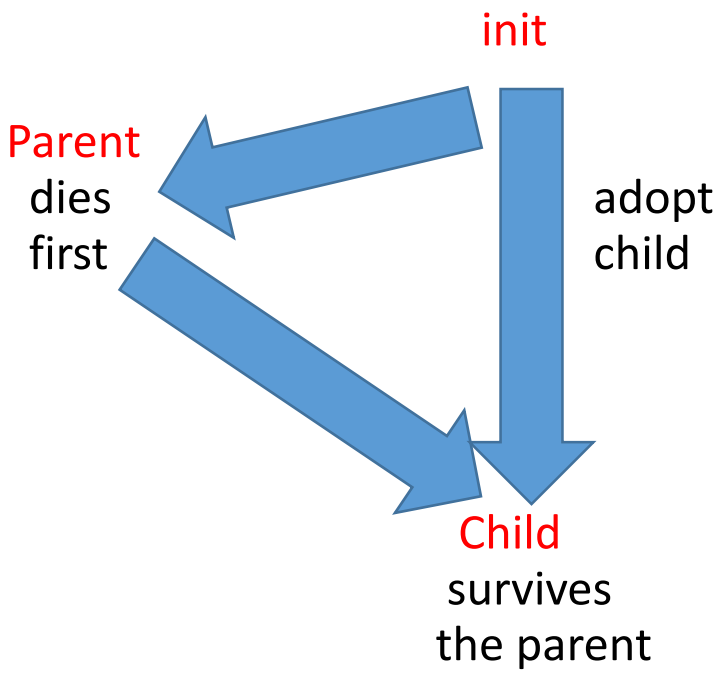
- Parents are responsible for reaping their children
- What should happen if parent terminates without reaping its children?
- Who reaps the children?

Orphaned Processes

- **Orphan: A process that has not been reaped by its terminated parent**
- **Orphaned processes are adopted by the OS kernel**
- **… and the kernel always reaps its children**

Orphan Process

- If parent process does not wait for child and it first terminates leaving child process orphan
  - Orphan processes are adopted by init process which started the parent (i.e. parent of parent)



Why study wait() system call?

System call
wait()
is useful
to avoid orphan processes, prevent race conditions, ensure process synchronization etc.

Concept:

Parent process should wait on its child processes.
Therefore, wait() call is mostly used in a parent process.

Program to demonstrate ZOMBIE processes

Terminal

zombie.c (~/Desktop/SS 2021/Wk-4/example code/wait) - gedit

Open Save Undo

zombie.c × race.c ×

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

main()
{
int pid;
pid = fork(); /* Duplicate */
 if ( pid!= 0 ) /* Branch based on return value from fork()*/
 { while(1) /* Never terminate, never execute a wait () */
   sleep(1000);
 }
 else {
  exit(84); /* Exit with a silly number */
 }
}
```

jayprakash@jayprakash-System-AsusPG500: ~

```
$ gcc zombie.c -o zombie
$ ./zombie  &
[1] 2457
$ ./zombie  &
[2] 2460
$ ./zombie  &
[3] 2463
$ ./zombie  &
[4] 2466
$
```

jayprakash@jayprakash-System-AsusPG500: ~

```
$ ps -a
  PID TTY          TIME CMD
 2457 pts/1    00:00:00 zombie
 2458 pts/1    00:00:00 zombie <defunct>
 2459 pts/8    00:00:00 ps
$ ps -a
  PID TTY          TIME CMD
 2457 pts/1    00:00:00 zombie
 2458 pts/1    00:00:00 zombie <defunct>
 2460 pts/1    00:00:00 zombie
 2461 pts/1    00:00:00 zombie <defunct>
 2462 pts/8    00:00:00 ps
$ ps -a
  PID TTY          TIME CMD
 2457 pts/1    00:00:00 zombie
 2458 pts/1    00:00:00 zombie <defunct>
 2460 pts/1    00:00:00 zombie
 2461 pts/1    00:00:00 zombie <defunct>
 2463 pts/1    00:00:00 zombie
 2464 pts/1    00:00:00 zombie <defunct>
 2465 pts/8    00:00:00 ps
$ ps -a
  PID TTY          TIME CMD
 2457 pts/1    00:00:00 zombie
 2458 pts/1    00:00:00 zombie <defunct>
 2460 pts/1    00:00:00 zombie
 2461 pts/1    00:00:00 zombie <defunct>
 2463 pts/1    00:00:00 zombie
 2464 pts/1    00:00:00 zombie <defunct>
 2466 pts/1    00:00:00 zombie
 2467 pts/1    00:00:00 zombie <defunct>
 2468 pts/8    00:00:00 ps
$
```

Program to demonstrate RACE CONDITION

Text Editor — race.c (~/Desktop/SS 2021/Wk-4/example code/wait) - gedit

zombie.c | race.c

```c
#include <stdio.h>
#include <unistd.h>

static void char_at_a_time(char *str) {
char *ptr;
int c;
  setbuf(stdout, NULL);
  for (ptr=str;c=*ptr++;)
     putc(c,stdout);
}

void main()
{
pid_t pid;

 if ((pid = fork()) < 0) printf("fork error!\n");
 else if (pid == 0)
        char_at_a_time("12345678901234567890\n");
 else
        char_at_a_time("abcdefghijklmnopqrstuvwxyz\n");
}
```

C | Tab Width: 8 | Ln 20, Col 1 | INS

jayprakash@jayprakash-System-AsusPG500: ~

```
$ gcc race.c -o race
$ ./race
abcdef12g3h4i5j6k7l8m9n0o1p2q3r4s5t6u7v8w9x0y
z
$ ./race
abcdefghijklmnopqrstuvwxyz
12345678901234567890
$ ./race
abcdefgh12i3j4k5l6m7n8o9p0q1r2s3t4u5v6w7x8y9z0

$ ./race
a12345678901234567890
bcdefghijklmnopqrstuvwxyz
$
```

Program to demonstrate use of wait() call to avoid RACE CONDITION

Terminal

race.c (~/Desktop/SS 2021/Wk-4/example code/wait) - gedit

Open Save Undo

zombie.c × race.c ×

```c
#include <stdio.h>
#include <unistd.h>

static void char_at_a_time(char *str) {
char *ptr;
int c;
  setbuf(stdout, NULL);
  for (ptr=str;c=*ptr++;)
     putc(c,stdout);
}

void main()
{
pid_t pid;

 if ((pid = fork()) < 0) printf("fork error!\n");
 else if (pid == 0)
        char_at_a_time("12345678901234567890\n");
 else
        {wait(NULL); char_at_a_time("abcdefghijklmnopqrstuvwxyz\n");}
}
```

C Tab Width: 8 Ln 20, Col 70 INS

jayprakash@jayprakash-System-AsusPG500: ~

```
$ gcc race.c -o race_2_norace
$ ./race_2_norace
12345678901234567890
abcdefghijklmnopqrstuvwxyz
$
$ ./race_2_norace
12345678901234567890
abcdefghijklmnopqrstuvwxyz
$
$ ./race_2_norace
12345678901234567890
abcdefghijklmnopqrstuvwxyz
$
```

Need to have parent wait for child or child wait for parent to complete the critical section (CR) code. This can be done using signals which we will discuss next.