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;
}
```

Reaping Children

- **wait(): parents reap their dead children**
  - Given info about why child died, exit status, etc.

- **Two variants**
  - wait(): wait for and reap next child to exit
  - waitpid(): wait for and reap specific child

`pid_t wait(int *stat_loc);`

when called by a process with >=1 children:

- ***waits*** **(if needed) for a child to terminate**
- ***reaps*** **a terminated child (if >= 1 terminated children, arbitrarily pick one)**
- ***returns*** **reaped child's pid and exit status info via pointer (if non-NULL)**

when called by a process with no children:

- **return -1 immediately**

Example program

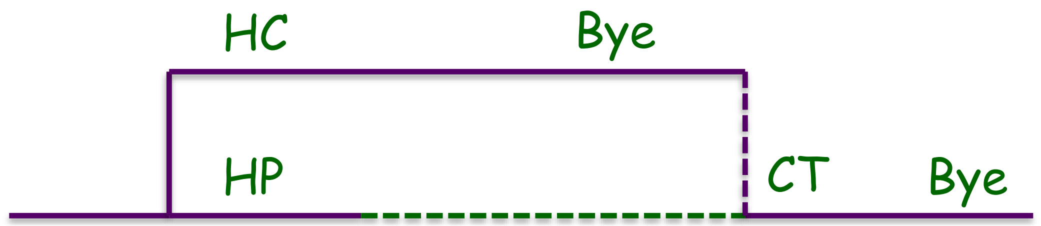
```
int main() {
   pid_t cpid;
   if (fork()== 0)
      exit(0);              /* terminate child */
   else
      cpid = wait(NULL); /* reaping parent */

   printf("Parent pid = %d\n", getpid());
   printf("Child pid = %d\n", cpid);

}
```

Example program

```
int main() {
    if (fork()== 0){
        printf("HC: hello from child\n");
    } else {
        printf("HP: hello from parent\n");
        wait(NULL);
        printf("CT: child has terminated\n");
    }
    printf("Bye\n");
}
```


HC
Bye
HP
CT
Bye