

Child status information

- **status information about the child reported by wait is more than just the exit status of the child**
 - normal/abnormal termination
 - termination cause
 - exit status

WIF... macros

- **WIFEXITED(status) : child exited normally**
 - **WEXITSTATUS(status) :** return code when child exits
- **WIFSIGNALED(status) : child exited because a signal was not caught**
 - **WTERMSIG(status) :** gives the number of the terminating signal
- **WIFSTOPPED(status) : child is stopped**
 - **WSTOPSIG(status) :** gives the number of the stop signal

`/* prints information about a signal */`

- **void psignal(unsigned sig, const char *s) ;**

```
void wait_x() {  
    int stat;  
    if (fork() == 0)  
        exit(1);  
    else  
        wait(&stat);  
    if (WIFEXITED(stat))  
        printf("Exit status: %d\n", WEXITSTATUS(stat));  
    else if (WIFSIGNALED(stat))  
        psignal(WTERMSIG(stat), "Exit signal");  
}
```

```
linux> ./w_x  
Exit status: 1
```

- If multiple children completed, will reap in arbitrary order

```
void wait_x() {
    int i, stat;
    pid_t pid[5];
    for (i=0; i<5; i++)
        if ((pid[i] = fork()) == 0){
            sleep(1);
            exit(100+i);
        }
    for (i=0; i<5; i++) {
        pid_t cpid = wait(&stat);
        if (WIFEXITED(stat))
            printf("Child %d terminated with status: %d\n",
                cpid, WEXITSTATUS(stat));
    }
}
```

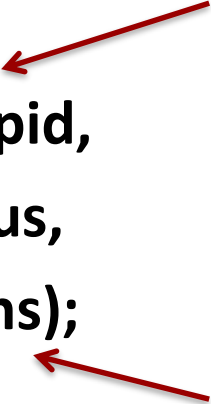
waitpid(): waiting for a specific process

- Useful when parent has more than one child, or you want to check for exited child but not block

```
pid_t result =  
    waitpid(child_pid,  
            &status,  
            options);
```

The child to wait for/check on
-1 means any child

0 = no options, wait until child exits
WNOHANG = don't wait, just check



■ Return value

- pid of child, if child has exited
- 0, if using WNOHANG and child hasn't exited

■ Can use waitpid() to reap in order

```
void wait_x() {
    int i, stat;
    pid_t pid[5];
    for (i=0; i<5; i++)
        if ((pid[i] = fork()) == 0){
            sleep(1);
            exit(100+i);
        }
    for (i=0; i<5; i++) {
        pid_t cpid = waitpid(pid[i], &stat, 0);
        if (WIFEXITED(stat))
            printf("Child %d terminated with status: %d\n",
                cpid, WEXITSTATUS(stat));
    }
}
```

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**