THE UNIVERSITY of NEW ORLEANS

DEPARTMENT OF COMPUTER SCIENCE

CSCI 2467, Fall 2019

Class Activity: graphing processes with signals

1 Using `fork()` with signals

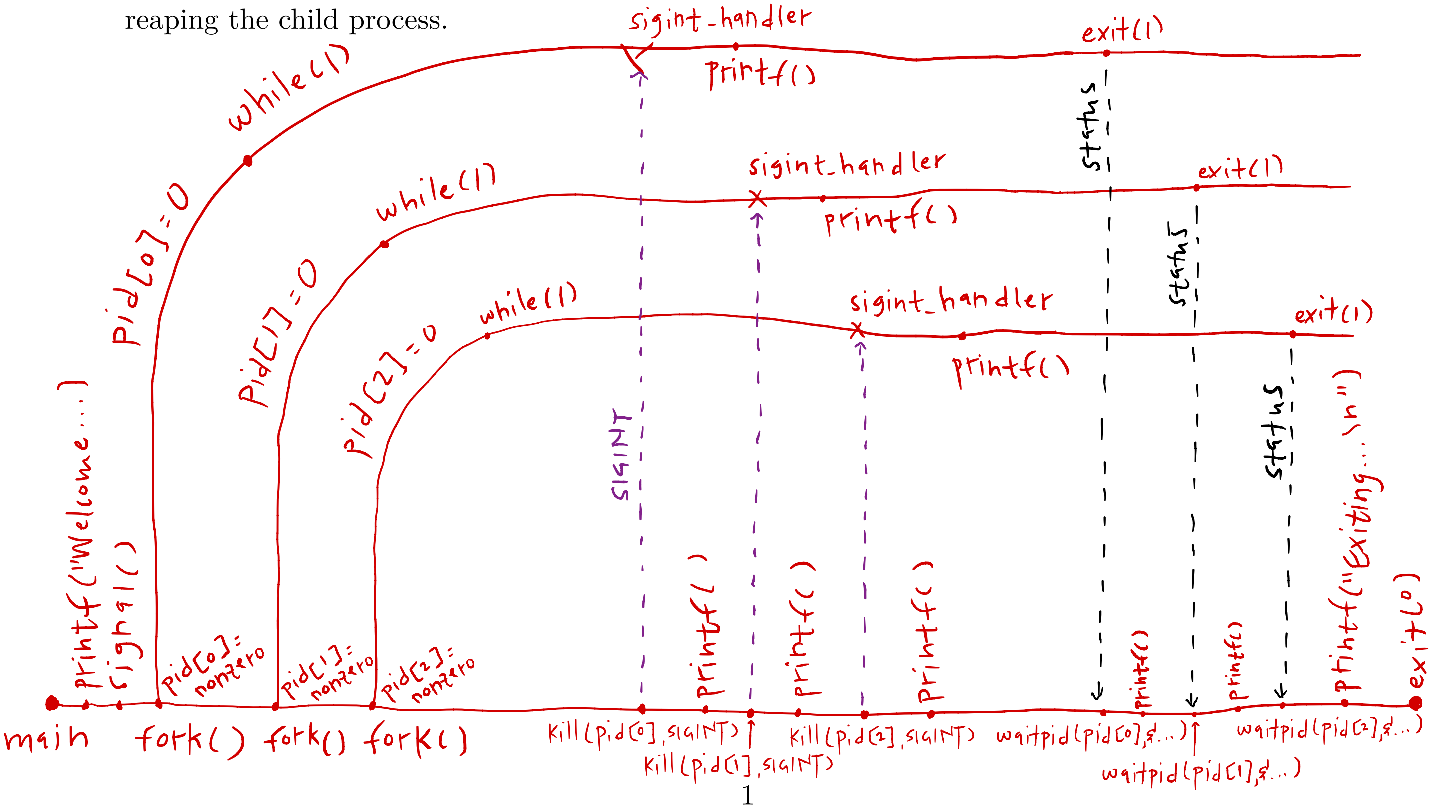
Consider the program on the next page, which contains a `main()` function and a signal handler called `sigint_handler` (The C source for this program is also available from the 2467 schedule page as usual, called `forkSig.c`.)

1.1 Commenting

Because this program is more complex than our last activity, you need to read and annotate the source code before making a process graph. On the lines that begin with `//`, answer the question given in the comments. (Comments using the `/* */` notation are already complete, and can help you understand the code.)

1.2 Graphing

Use the space below to draw a process graph for this program. Make sure that `fork()` calls are shown as a new branch in the graph. The `kill()` calls should be shown sending a signal to another process by using an arrow. The `wait()` or `waitpid()` calls should also use an arrow to show them reaping the child process.



```
void sigint_handler(int sig)
{
    printf("Process %d received signal %d\n", getpid(), sig);
    exit(1); /* set exit status 1 and end process */
}

int main()
{
    printf("\nWelcome to forkSig, a signal handling example!\n\n");

    int N = 3;
    pid_t pid[N];
    int child_status;

    // What does this signal() function call do?
    signal(SIGINT, sigint_handler);
```

registers sigint_handler as the signal handler for the SIGINT signal

```
    /* Create N processes and store their pids in the pid[] */
    for (int i = 0; i < N; i++) {
    // What's going into pid[i] here?
        pid[i] = fork();
        if (pid[i] == 0) {
        /* If you're the child, go into an Infinite Loop */
            while(1);
        }
    }
```

in child process, pid[i] will be ∅

in parent process, pid[i] will be pid of child (nonzero)

```
    for (int i = 0; i < N; i++) {
    /* signal each of the N processes referenced in the pid[] */
        printf("Sending SIGINT to process %d\n", pid[i]);
    // What is happening with this kill() function call?
        kill(pid[i], SIGINT);
    }
```

parent process sends SIGINT signal to child process

child process receives SIGINT, calls handler (breaks out of infinite loop)

```
    for (int i = 0; i < N; i++) {
    /* Reap each of the child processes */
        pid_t wpid = waitpid(pid[i], &child_status, 0);
    // What is the relationship between WIFEXITED and WEXITSTATUS?
        if (WIFEXITED(child_status))
            printf("Child %d terminated with exit status %d\n",
                    wpid, WEXITSTATUS(child_status));
        else
            printf("Child %d terminated abnormally\n", wpid);
    }
    printf("\nExiting...\n");
    exit(0);
}
```

- which signal?

↑ what was argument to child's exit() call?