# CISC 3350
# Workstation Programming
Dayton Clark
Spring 2013

# `forkwait` Assignment

## Introduction

In this assignment you will use the fork() and wait() system calls. You are to write a C/C++ program called, `forkwait.c`.

## What is to be done?

I will provide you with an outline of the program. The outline includes a couple of useful functions (`elapsedTime()` and `generateSeed()`) and some important constants for your program (`MINFORKS`, `MAXFORKS`, `MINPAUSE`, and `MAXPAUSE`).

You are to write a C/C++ program which behaves as follows:

1. Generate a random integer between `MINFORKS` and `MAXFORKS`. This is the number of child processes that to be forked. Use the library function `rand_r(3)` to generate random numbers.

2. Before each process is forked, a random integer between `MINPAUSE` and `MAXPAUSE`. This is the number of micro-seconds that the new process will pause before exiting. Note that the number for the length of pause *must* be generated in the main process not within the child process. Why?

3. Each child process is given a `childid` (0 <= `childid < forks`) by the main process. Note `childid` has nothing to do with the the child process' `pid` but is a small integer.

4. Each child process simply sleeps for the indicate time (use `usleep(3)`) and then exit with the exit status equal to the process' `childid`.

5. The parent process, after creating all the child processes, waits for each of the child processes. The children should be captured in the order they terminate, not the order they were created. When a process terminates the parent process gets the child's `childid` from the exit status and prints it.

One run of my version of the program produced the following output:

```
 0: Start
17: Number of forks = 7
```

```
     112: child  0, pid = 21819 forked, waits 1067330 usec
     180: child  1, pid = 21820 forked, waits  201668 usec
     248: child  2, pid = 21821 forked, waits  330457 usec
     318: child  3, pid = 21822 forked, waits 1077631 usec
     386: child  4, pid = 21823 forked, waits 1696229 usec
     454: child  5, pid = 21824 forked, waits  664332 usec
     524: child  6, pid = 21825 forked, waits  134828 usec
  136434: child  6, pid = 21825 terminated
  202807: child  1, pid = 21820 terminated
  331677: child  2, pid = 21821 terminated
  665758: child  5, pid = 21824 terminated
 1068415: child  0, pid = 21819 terminated
 1078849: child  3, pid = 21822 terminated
 1697627: child  4, pid = 21823 terminated
 1697666: End
```

The numbers to the left are the time, in micro-seconds, since the beginning of the parent process. The provided function, `elapsedTime()`, returns the appropriate time. The first time `elapsedTime()` is called it should be called with an argument of `0`. Subsequently it must be called with a non-zero argument.

Your program must produce output similar to the above. **Your output must be at least as informative and at least as pleasant to look at as the above**.

## Random Numbers

Use the function `rand_r(3)` to generate random numbers. `rand_r()` takes an argument of `(unsigned *)` which is the address of the "seed". Whatever seed variable you use, initialize it with `generateSeed(0)`. This function generates a seed based on the current time.

## Files

I provide three files from the Sakai site for the course for this assignment. They are `Makefile`, `forkwait.c`, and `forkwait.pdf` (this file).

### ***`Makefile`***

You do not have to use this file, but it might your work a little easier. If you want to use this file then this file must be in the same directory as your source file, forkwait`.c`. To use it simply type:

```
> make test
```

It will compile your program, and if there are no errors, run your program. So if your program has no errors it will look something like this:

```
> make test
cc -g -o forkwait forkwait.c
./forkwait
        0: Start
       18: Number of forks = 8
      105: child  0, pid = 61456 forked, waits 1983946 usec
      177: child  1, pid = 61457 forked, waits  765598 usec
      248: child  2, pid = 61458 forked, waits  402627 usec
      317: child  3, pid = 61459 forked, waits 1844170 usec
      389: child  4, pid = 61460 forked, waits  820717 usec
      461: child  5, pid = 61461 forked, waits   63232 usec
      532: child  6, pid = 61462 forked, waits 1804350 usec
      602: child  7, pid = 61463 forked, waits 1474254 usec
    64832: child  5, pid = 61461 terminated
   404129: child  2, pid = 61458 terminated
   766765: child  1, pid = 61457 terminated
   822222: child  4, pid = 61460 terminated
  1476017: child  7, pid = 61463 terminated
  1806088: child  6, pid = 61462 terminated
  1845445: child  3, pid = 61459 terminated
  1985133: child  0, pid = 61456 terminated
  1985288: End
```

In addition, if you are online, then you can also test your program on one of the Solaris machines in the WEB building as follows:

```
make itstest
ssh dayton@fldsun250.its.brooklyn.cuny.edu "mkdir forkwait; chmod 0700 forkwait"
Access to Brooklyn College computers is restricted to authorized users
       and approved educational and research purposes, only.
mkdir: Failed to make directory "forkwait"; File exists
scp Makefile forkwait.c dayton@fldsun250.its.brooklyn.cuny.edu:forkwait
Access to Brooklyn College computers is restricted to authorized users
       and approved educational and research purposes, only.
Makefile                                                100% 1862     1.8KB/s   00:00
forkwait.c                                              100% 3432     3.4KB/s   00:00
ssh dayton@fldsun250.its.brooklyn.cuny.edu "cd forkwait; /usr/ccs/bin/make test"
Access to Brooklyn College computers is restricted to authorized users
       and approved educational and research purposes, only.
cc -g -o forkwait forkwait.c
./forkwait
        0: Start
     1458: Number of forks = 9
     4434: child  0, pid = 92461 forked, waits   37372 usec
     7273: child  1, pid = 92462 forked, waits   37072 usec
    10061: child  2, pid = 92463 forked, waits   42552 usec
    13139: child  3, pid = 92464 forked, waits   23614 usec
    14474: child  4, pid = 92465 forked, waits   37246 usec
    15783: child  5, pid = 92466 forked, waits   22047 usec
    17048: child  6, pid = 92467 forked, waits   13797 usec
    18313: child  7, pid = 92468 forked, waits   26674 usec
    19568: child  8, pid = 92469 forked, waits   41579 usec
    37803: child  3, pid = 92464 terminated
    39406: child  6, pid = 92467 terminated
    42739: child  0, pid = 92461 terminated
    45289: child  1, pid = 92462 terminated
    46597: child  5, pid = 92466 terminated
    54581: child  7, pid = 92468 terminated
    55102: child  2, pid = 92463 terminated
    60212: child  4, pid = 92465 terminated
    69411: child  8, pid = 92469 terminated
    69712: End
```

Note that the `mkdir` command failed in the fifth line, this is normal. Also note, again, that if you are working on one of the ITS machines in the WEB building, you do not need to use `make itstest`.

I would like you to get your program to work in both your preferred local environment and on the ITS/WEB machines.

### ***`forkwait.c`***

This is the outline or shell of the program you are to write. It includes everything except the body of `main()`, which of course, is the most important part.

### ***`forkwait.pdf`***

`This file is the document you are reading in PDF format. Besides the description of the assignment this file has some question for you to consider.`

## Style

One of the themes of this course is to teach how to write good programs. So your program for this assignment and all assignments must reflect this. Your program should have the following properties:

- The program must run correctly.
- The output of the program should be clear and easy to understand and pleasant to look at.
- The program code should be clear and easy to understand
- The code must be indented.
- Comments in the source file must be clear and easy to understand and pleasant to look at.
- You must check for and handle errors from system calls and library functions.
- You must explicitly exit from the program with an appropriate exit code.

## Cooperation

I encourage you to discuss the assignment with your fellow students. However, you must each turn in individual, distinct, and different versions of the program.

## What to submit

You must submit 2 files via the Sakai site for the course. They are:

1. Your source code for the assignment, `forkwait.c`.

2. The output from your program run *three* separate times

## Sakai

This description of the assignment is available on the BC Sakai site: https://learn.brooklyn.edu/xsl-portal. You must submit the two files described above to the site by the due date. As described above.

Note: The files must be submitted as submissions for this assignment, *not* via the Sakai dropbox.

## Questions

Here are two questions for you to consider. You need *not* submit an answer now for them. But consider them, they may come up in class or on a test or while you are working on the program.

Q1. When running remotely on the ITS system (i.e. using `make itstest`) I would get output like

```
./forkwait
         0: Start
      1055: Number of forks = 4
      3765: child  0, pid = 93087 forked, waits   31937 usec
      6288: child  1, pid = 93088 forked, waits   26159 usec
         0: Start
      1055: Number of forks = 4
      3765: child  0, pid = 93087 forked, waits   31937 usec
         0: Start
      1055: Number of forks = 4
         0: Start
      1055: Number of forks = 4
      3765: child  0, pid = 93087 forked, waits   31937 usec
      6288: child  1, pid = 93088 forked, waits   26159 usec
      8608: child  2, pid = 93089 forked, waits   16849 usec
         0: Start
      1055: Number of forks = 4
      3765: child  0, pid = 93087 forked, waits   31937 usec
      6288: child  1, pid = 93088 forked, waits   26159 usec
      8608: child  2, pid = 93089 forked, waits   16849 usec
     10926: child  3, pid = 93090 forked, waits   32315 usec
     26946: child  2, pid = 93089 terminated
     33734: child  1, pid = 93088 terminated
     36901: child  0, pid = 93087 terminated
     44426: child  3, pid = 93090 terminated
     44438: End
```

I fixed the problem by putting "`setbuffer(stdout, NULL, 0);`" as the first statement in my program. What was causing the problem and why did this fix it?

Q2. In my description of the program I state "Note that the number for the length of pause *must* be generated in the main process not within the child process." Why is this so? What might happen if the length of the wait were generated in the child process?