Homework 3
Operating Systems, V22.0202
Fall 2007, Professor Yap

Due: Wed Oct 10

- Please read questions carefully. When in doubt, please ask.
- The written homework is to be submitted in hardcopy during class, but the programming part sent to us in a single file (as detailed below) by midnite.

Question 1 (5 Points)
Problem 4.2, p.146. How to do context switch between user-level threads. ♢

Question 2 (5 Points)
Problem 4.5, p.147. Multithreaded soluting using user-level threads on a multiprocessor versus uniprocessor. Note: you must give the reasoning behind your "Yes" or "No" answer. ♢

Question 3 (15 Points)
In Homework 2, we implemented the toy shell. Suppose we want to implement pipes between processes. Describe what changes needs to be done to your `tsh.c` program. For simplicity, assume that there is only one use of the pipe directive "|".

HINT: you need to use the `dup()` system call. You need not write out complete programs, but give explicit code fragments to show how the critical parts are implemented. ♢

Question 4 (70 Points)

- INTRODUCTION. In this programming homework, you are to simulate the concurrent solution of a computational task using processes.

  The task is to to compute the GCD (greatest common divisor) of one or more pairs of numbers. Given two positive integers m and n, their $GCD(m, n)$ is defined to be the largest number that divides both m and n. Clearly, $GCD(m, n) \geq 1$. For instance, $GCD(15, 9) = 3$. There is the well-known Euclidean algorithm for computing GCD. Since $GCD(m, n) = GCD(n, m)$, we may assume that $m \geq n > 0$. Initially, let

  $$m_0 = m, \qquad m_1 = n.$$

  Then Euclid's algorithm says to compute the sequence of integers,

  $$(m_0, m_1, m_2, \ldots, m_k, 0) \tag{1}$$

  where for each $i \geq 2$, we define $m_{i+1} = m_{i-1} \mod m_i$. Recall that the **modulo operation** $a \mod b$ simply returns the remainder of a divided b. Hence $0 \leq (a \mod b) < b$. Thus $0 \leq m_{i+1} < m_i$ for $i \geq 2$. Hence the sequence (1) must eventually reach 0. If $m_{k+1} = 0$ (for some $k \geq 1$) then it is easy to show that the previous number m_k is the GCD.

  E.g., for $(m, n) = (15, 24)$, we get the sequence $(24, 15, 9, 6, 3, 0)$ and so $k = 4$ and $m_4 = \texttt{GCD}(15, 9) = 3$. A sample program `gcd.c` for computing gcd is provided here.

- PROBLEM OVERVIEW. Your main program should be called `mgcd.c` (multiple gcd). The input to `mgcd.c` is a sequence of pairs of numbers. If there are $k \geq 1$ numbers, then the main process will spawn $\lfloor k/2 \rfloor$ children processes. Each child process will compute the GCD of one pair of numbers. However, the child process does not know how to compute the modulo operation. Only the parent process knows how. Hence the child must submit pairs (a, b) of integers to the parent who will compute and return the modulus $a \mod b$. When a child process has computed the GCD of its pair, it exits. When all the children has exited, the parent exit. For instance, if we type

```
> gcc mgcd.c -o mgcd
> mgcd 24 17 18 10 987654321 123456
```

then GCD will spawn 3 processes which eventually prints "1", "2" and "3" (these are the GCD's of the three pairs).

We want the parent process to implement the $a \mod b$ operation by repeated subtractions. Moreover, the parent should use round robin method to service the children.

- COMMUNICATION. The parent and each child communicates through a **two-way pipe** (this is just two pipes, one from parent-to-child, and another from child-to-parent). That is, the child will write a pair (a, b) of integers in the child-to-parent pipe, and the parent will respond by writing the integer $(a \mod b)$ in the parent-to-child pipe. There are two ways to do this:
  (a) One way is for the parent to try to read from each of its "children-to-parent" pipes in the round-robin fashion. To accomplish this, the parent must perform what is known as a **nonblocking read**. Such a read will never block – even if there is nothing to read. The other kind of reading is called a **blocking read**. Note that it is quite acceptable for the child to read the parent-to-child pipe in a blocking manner.
  (b) The other way is to use the **signaling mechanism** for unix processes. A child can signal the parent after it has placed a pair (a, b) in the child-to-parent pipe. The parent responds to the signal by searching through the pipes from each of the children to find a pair (a, b) to work on. In order to avoid busy waiting, the parent will only do this search in response to a signal.

  BUT NOTE THAT WE REQUIRE the non-signaling version (i.e., method (a)) for this homework.
- Finally, you are to create several runs using various sets of input pairs, and give the timing for these runs.
- Thus, you must know how to:
  - set up pipes
  - how to read/write from and to pipes
  - how to do non-blocking reads
  - time the running time of your program.

  We will give you all the hints necessary to do the programming part.
- WHAT TO HAND IN: similar to previous homeworks, we want a single tar file containing a Makefile file, README file, and all necessary programs. You must give your timings and explain your experiments in the README file. We should be able to duplicate your experiments by typing "make time".
- USEFUL INFORMATION.
  - To read an integer from the command line, it is useful to know the `atoi()` library function to convert a string of digits into an integer. E.g., the following code fragment converts the first two arguments of the command line into integers:

    ```
    int arg1 = atoi(argv[1]);
    int arg2 = atoi(argv[2]);
    ```
  - Here is a routine to set the status flag for files or pipes.

    ```
    #include <stdio.h>
    #include <sys/types.h>
    #include <errno.h>
    #include <stdlib.h>
    #include <string.h>        // include all these for convenience

    void setFlag(int fd, int flags) {
                                // flag is a bitvector for bits to turn on
      int val;
      if ((val = fcntl(fd, F_GETFL, 0)) < 0)  // get original bits
    ```

```
     perror("fcntl F_GETFL error");     // F_GETFL are predefined
   val |= flags;                             // turn on the bits in flag;
   if (fcntl(fd, F_SETFL, val) <0)           // set the new bits
     perror("fcntl F_GETFL error");
}
```

For the parent to read the "child2parent pipe" in a nonblocking way, we execute:

```
setFlag( child2parent[0], O_NONBLOCK);     // O_NONBLOCK are predefined
```

– How do you send a pair of integers, m and n, to the parent on the pipe? Here is a solution sprintf and sscanf.

1. Child writes the values of m and n in buffer:
   `sprintf(buf,"%d,%d", m, n);`
2. Child writes buf into the pipe to the parent.
3. Parent reads from pipe into its own BUF.
4. Parent use sscanf to decode from BUF:
   `sscanf(BUF,"%d,%d", &M, &N);`
   where M, N are integers to hold the values of m and n.

♢