ECE 434: Introduction to Computer Systems (or Operating Systems), Spring 2017

Instructor: Maria Striki **03-17-2017**

TA: Musaab Alaziz

Graders: Christina Segerholm, Ashish Behl

Homework 2 (150 points), 30 each HW problem

Due Date: Tue April 4th 2017 in class (with a week extension possibility)

Problem 1:

The Sieve of Eratosthenes is a simple, ancient algorithm for finding all prime numbers up to any given limit. It does so by iteratively marking as composite (i.e., not prime) the multiples of each prime, starting with the multiples of 2.

https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

The sieve of Eratosthenes can be expressed in pseudocode, as follows:

```
Input: an integer n > 1.

Let A be an array of Boolean values, indexed by integers 2 to n,
initially all set to true.

for i = 2, 3, 4, ..., not exceeding √n:
  if A[i] is true:
    for j = i², i²+i, i²+2i, i²+3i, ..., not exceeding n:
      A[j] := false.

Output: all i such that A[i] is true.
```

This algorithm produces all primes not greater than n. It includes a common optimization, which is to start enumerating the multiples of each prime i from i^2. The time complexity of this algorithm is $O(n \log \log n)$.

What to do: Write a multi-threaded program (using C and Pthreads library) that outputs prime numbers. The program should work as follows: the user will run the program and enter a

number on the command line. The program creates a thread that outputs all the prime numbers less than or equal to the number entered by the user.

Also, create a separate thread that outputs this subset of the above primes that have the following property: the number that is derived by reversing the digits is also prime (e.g., 79 and 97).

Problem 2:

i) Draw the graph processes in its final form, i.e., when all processes and threads have reached a steady state. Highlight processes with a circular node and the threads with a square node in the graph. For each node make sure that the parental process appears with an arrow. Also, show the process/threads who have lived temporarily but they do not exist in the steady state, by means of a broken outline.

ii) For each node in the graph indicate: a) the system call or function within the Program Counter of the corresponding process / thread located is, b) the arguments with which this function has been called, and c) the program line where this call took place. For the processes / threads of temporary life, highlight last system call or function within which the Program Counter was found.

iii) Complete your shape to show the inter-process communication: for each data transfer or signal delivery , design a dashed arrow from the sender-to-recipient process. Above the arrow type the value transferred each time.

iv) What will be printed in lines 54 and 56 of the program?

Note: From the man page of kill we're given: int kill (pid_t pid, int sig);

If pid equals 0, then sig is sent to every process in the process group of the calling process.

```c
int res = 0, num = 2, pd[2], arg[32];
pthread_t thread[32];
sem_t sem;

void h(int s)
{
    read(pd[0], &num, sizeof(num));
    sem_post(&sem);
}

void *f(void *var)
{
    int tmp = *(int *)var;
    res += tmp;
    pthread_exit((void *)0);
}

int main()
{
    int i, k, p, pstatus;
    void *tstatus;

    sem_init(&sem, 0, 0);
    signal(SIGUSR1, h);
    pipe(pd);

    for (i = 0; i < num; i++) {
        p = fork();

        if (p == 0) {
            sem_wait(&sem);
            num++;
            for (k = 0; k < num; k++) {
                arg[k] = k;
                pthread_create(&thread[k], NULL, f, (void *)&arg[k]);
            }

            for (k = 0; k < num; k++) pthread_join(thread[k], &tstatus);
            if (num < 4) write(pd[1], &num, sizeof(num));

            exit(res);
        }
    }

    write(pd[1], &num, sizeof(num));
    signal(SIGUSR1, SIG_IGN); /* Το SIGUSR1 αγνοείται από την καλούσα διεργασία */
    kill(0, SIGUSR1);   /* Δείτε επισήμανση στην εκφώνηση */

    for (i = 0; i < num; i++) {
        wait(&pstatus);
        if (WIFEXITED(pstatus)) res += WEXITSTATUS(pstatus);
    }

    printf("Final result1: %d \n", res);
    read(pd[0], &num, sizeof(num));
    printf("Final result2: %d \n", num);

    return 0;
}
```

Problem 3: Bank Simulation

i) What are semaphores and what is its usefulness? What type are the synchronization problems that a semaphore may resolve but the simple lock cannot?

ii) Provide the definition of a semaphore by Dijkstra. The implementation based on this definition leads to the issue of "busy waiting". What is this problem and why is it undesirable?

iii) Provide an implementation of the semaphore that avoids the busy wait. In which places is now the waitig restricted?

iv) Implement a synchronization scheme that simulates the behavior of the customers of a bank as follows: The bank acquires K seats in the waiting room and a customer service desk. The customers can see from the window if there is free seating. If there is not, they go for a walk (take_a_walk ()) and retry later. If there is seating available, they enter the waiting room and attempt to be served one at a time in the customer service desk. The client is served by calling *make_transaction ()*. Use shared variables and semaphores for your solution. Consider making any changes necessary at the points indicated with. … in the code segment that follows:

```
. . .
void bank_client()
{
    while (1) {
        . . .
        if (. . .) { /* if seats available */
            . . .
            make_transaction();
            . . .
            break;
        }
        else {
            . . .
            take_a_walk();
        }
    }
    return_home();
}
```

Problem 4: The "Kindergarten" Problem

Facing the kindergarten synchronization problem ("Kindergarten"), where regulations require that one teacher per R children is always present, where R is an integer (e.g., child / teacher ratio 3: 1) to ensure proper supervision of children.

Moreover, we would wish for our configuration to have the following characteristics:

1. If a teacher attempts to leave but this is not possible, he should returns in office (and not block while waiting for the exit condition to be met).

2. Every parent should be able to enter the nursery area to verify whether the regulation is met.

Please provide the synchronized processes: Teacher, Children, Parents, which satisfy the requirements above, by further defining functions:

_enter (), _exit () and verify_compliance () that are subsequently used, or replace them (their call) with the appropriate code within: Teachers, Children and Parents. You may use mutexes/ locks, semaphores, shared variables).

```
void Teacher()
{
    for (;;) {
        teacher_enter();
        ...critical section...
        teach();
        teacher_exit();
        go_home();
    }
}
```

```
void Child()
{
    for (;;) {
        child_enter();
        ...critical section...
        learn();
        child_exit();
        go_home();
    }
}
```

```
void Parent()
{
    for (;;) {
        parent_enter();
        ...critical section...
        verify_compliance();
        parent_exit();
        go_home();
    }
}
```

Problem 5:

A C program in UNIX environment is given below the questions.

Consider that the system calls do not fail, function Fn () never returns, Fn () does not execute system calls, each new process inherits exactly the way signals are handled by the parent process, and finally system calls can be interrupted by incoming signals. Since Fn () never returns, the processes that the program creates come to a permanent state: the process tree remains stable forever and each processs is in a specific function or system call.

a) Answer briefly the following questions:

1. What do the following calls do in UNIX: kill (pid, SIGSTOP) and signal (SIGINT, handler)?

2. What does in the call n = read (fd, buf, cnt) the argument cnt signify? What values can the variable n have after its return from the read ();

3. What does the call wp = wait (& status) do; Suppose that after calling the process with PID 1000 the following apply: wp == 1011, WIFSIGNALED (status) == 1, WTERMSIG (status) == 3. Assuming that process 1000 determines process 1011, what happens to process 1011 and with what call has process 1000 affected it?

4. What happens to a process that calls the read () in an empty pipe when the write end remains open (for writing)?

b) Draw the process tree in its final form, i.e., when all processes have reached a steady state. Explain briefly how it arises. Solve potential race condition in line 18, assuming that at least three processes arrive at the point marked / * A * /, passing the read () in the order of their birth.

c) At each node of the tree process please highlight: (i) the system call or a function where the Program Counter of the corresponding process is found, (ii) the arguments with which the process has been called, (iii) the program line from which the call took place.

For the arguments, make any assumptions you need to for the numbers you do not know, for example, the PIDs assigned by the system to the new processes.

d) Complete the process tree to show the inter-process communication: for each data transfer or signal sending that has occurred, draw a dotted arrow from the sender process to recipient process. Above the arrow type the value that is transferred each time.

```
1   int pg[2];
2
3   void handler(int signum)
4   { exit(6); Fn(0, -1); }
5
6   int main(void)
7   {
8       int i, j, n;
9       char c[10];
10      pid_t pid[4];
11
12      signal(SIGUSR1, handler);
13
14      pipe(pg);
15      for (i = 0; i < 4; i++) {
16          pid[i] = fork();
17          if (pid[i] == 0) {
18                  n = read(pg[0], c, i + 2);
19                  /* A */
20                  for (j = 0; j < n; j++) {
21                      pid[i] = fork();
22                      if (pid[i] == 0)
23                          Fn(getpid(), j);
24                  }
25                  for (j = 0; j < i; j++)
26                      wait(NULL);
27                  Fn(i, 12);
28                  exit(2);
29          }
30      }
31
32      kill(pid[i - 2], SIGUSR1);
33      for (i = 0; i < 4; i++) {
34          wait(&n);
35          write(pg[1], c, WEXITSTATUS(n));
36      }
37      Fn(i, -1);
38      return 0;
39  }
```