EEL 4732/5733 Advanced Systems Programming

Exam1

Student Name:

UFID:

Signature:

Important Note There are five questions. Please read all the questions carefully and write your name on all answer pages. Good luck.

Questions

1. (6 pts) For the variables x and y, show their locations in the virtual address space of each process that executes the program below along with their values before the `fork` system call, immediately after the `fork` system call, and just before the processes terminate. Also, show a possible output of the program as it will be displayed on the terminal assuming that the `fork` system call does not return -1.

```
1  static int x = 111;
2
3  int
4  main(int argc, char *argv[])
5  {
6      int y = 222;
7      pid_t result;
8
9      switch (result = fork()) {
10     case -1:
11         exit(1);
12
13     case 0:
14         x *= 3;
15         y *= 3;
16         break;
17
18     default:
19         break;
20     }
21
22     printf("PID=%ld x=%d iy=%d\n", (long) getpid(), x, y);
23
24     exit(0);
25 }
```

2. (6 pts) This question has two parts:

   (a) (3 pts) For each of the statements about the `exec` family of system calls, decide whether True or False:

      i. If `exec` succeeds, the virtual address space of the process gets replaced with a new one such that the text segment, the data segment, and the stack segment get initialized according to the program that gets `exec`'ed.

      ii. If `exec` succeeds, the file descriptor table is reinitialized such that it only has three valid entries: indices 0, 1, and 2 initialized with the standard input, standard output, and standard error respectively.

      iii. If `exec` succeeds, it does not return to the calling context, i.e., the statement that follows the `exec` call does not get executed.

      iv. If `exec` succeeds, the installed signal handlers get preserved.

      v. If `exec` succeeds, the pending signals get preserved.

      vi. If `exec` succeeds in a child process, it does not affect the parent process.

   (b) (3 pts) For each of the statements about the `wait`/`waitpid` system calls, decide whether True or False:

      i. The parent waits for the termination of any child with the `wait` system call.

      ii. If a process terminates and the parent is not currently waiting for it then the child becomes a zombie and stays as one until the parent waits for it.

      iii. If a parent process blocks on the `wait` system call then there must be a not yet terminated child process and no terminated child processes since the last call to `wait`.

      iv. Even though a parent process uses the `wait` system call, it cannot find out the termination status of the child process.

      v. A parent process can wait for a specific child by passing the process id of that child to the `waitpid` system call.

      vi. If `wait` is called within a handler for the SIGCHLD signal then this system call will not block and will return immediately.

Pending signal is not Environment variable

SIGCHILD signal means status changed, but is it equal to 'child terminate'? Please assume that.

3. (6 pts) Write the pseudocode for a program that creates a child process to communicate via a pipe, which will be used to let the parent process know about the termination of the child and its termination status. Each process will only print a specific string on the standard output before terminating: the parent prints "PARENT" and the child prints "CHILD". The child process calls a function `int mystery(void)` and uses that return value as the termination status. Note that you are not allowed to use the `wait` or `waitpid` system calls. To get full credit, you need to use all the necessary system calls in the right order, the processes should not block indefinitely, and no signals should be generated due to the incorrect usage of the system calls/data structures. Also, make sure that when the parent concludes that the child must have terminated, the child has really terminated.

```
pipe(fd);
x = fork();
switch(x){
   case -1: errExit("fork");
   case 0:
       close(fd[0]);
       status = mystery();
       write(fd[1], &status, sizeof(status));
       printf("CHILD");
       return 0;
default:
       close(fd[1]);
       while(read(fd[0], buffer, size)!=0)
       printf("PARENT");
       return 0;
}
```

4. (6 pts) Implement a solution to the producer consumer problem, where there is only one producer that produces finite amount of data and distributes the items to K consumer threads in a round robin way. Note that the consumer does not know the amount of data and so the producer has to let the consumers know that there won't be any more data. Each consumer thread has a separate data buffer of size M. Your pseudocode solution should be free of data races and deadlocks. Make sure to specify the initial states of your synchronization primitives if applicable.

5. (6 pts) This question has two parts:

   (a) In the code snippets below, the main thread intends to pass each thread a unique id value to be printed by that thread. For each of the following, explain whether it is the right way to pass the thread identifier as a parameter to the thread function `foo`, if it is defined as follows for both cases.

```
1  void *foo(void *arg) {
2     int *id = (int*)arg;
3     printf("thread %d\n", *id);
4     return 0;
5  }
```

   i. First approach:

```
1
2   pid_t pid[NUM};
3
4   int main(...) {
5        for(int i=0; i< NUM; i++) {
6            pthread_create(&pid[i], NULL, func, (void *)&i);
7        }
8        ...
9        for(int i=0; i< NUM; i++) {
10           pthread_join(pid[i], NULL);
11       }
12       return 0;
13  }
```

   ii. Second approach:

```
1
2   pid_t pid[NUM};
3   int tid[NUM];
4
5   int main(...) {
6        for(int i=0; i< NUM; i++) {
7            tid[i] = i;
8            pthread_create(&pid[i], NULL, func, (void *)&tid[i]);
9        }
10       ...
11       for(int i=0; i< NUM; i++) {
12           pthread_join(pid[i], NULL);
13       }
14       return 0;
15  }
```

(b) Assume that there is a library function `foo` that writes to a global data structure. The library developers have decided to rewrite `foo` to make it thread safe. Suggest three different mechanisms from the Pthreads library to implement a thread safe version of `foo`.

1、Use synchronization primitives
2、Use the thread specific data mechanism
3、Use thread local storage