

[Operating Systems - Fall 2018](#) >  [Assignments](#)

 [Assignments](#)

Assignments

Assignment - In progress

Complete the form, then choose the appropriate button at the bottom.

Title

Assignment 04 - Signals

Due

Nov 4, 2018 11:55 pm

Number of resubmissions allowed

Unlimited

Accept Resubmission Until

Nov 4, 2018 11:55 pm

Status

In progress

Modified by instructor

Oct 26, 2018 11:39 am

Instructions

Assignment 04 - Signals

Question 1 - Home brew of the sleep() call

The function `unsigned int sleep (int sec)` suspends the calling process until `sec` seconds have elapsed, or until a signal is delivered. The function returns 0 if the initial sleep time has fully elapsed, or the number of seconds remaining in the case of a delivered signal.

Program a function `unsigned int mysleep(int sec)` that uses the `alarm` system call to simulate the behavior of the `sleep` call.

NB: similarly to the POSIX specification of the `sleep` call, you can assume that the user will not call `alarm()` nor use the `SIGALRM` signal in the same program.

NB2: if your function alters the default mask and / or behavior associated with a signal received during a given period of time, you must restore the original mask and / or behavior afterwards.

Question 2 - Signal waves

We want to create a program `waves` that sends waves of signals along a lineage of processes. This program works as follows:

- The parent process PP associated with the main program creates a line of N processes (parent creates a child, child creates a grandchild, and so on ...)
- Each process, with the exception of the last descendant LD (ie. the last created process) blocks in wait of a `SIGUSR1`. Process LD will start a wave of `SIGUSR1` signals: it sends a `SIGUSR1` to its parent, which transmits it to its own parent, and so on until PP delivers the `SIGUSR1` received from its own child.
- At the end of the first wave, PP initiates a second wave with `SIGUSR2` signals. This second wave stops when it reaches LD.
- Once it receives the `SIGUSR2` signal of the second wave, LD initiates a third and last wave with `SIGINT` signals. This last wave is different from the previous ones: every child process terminates after emitting a `SIGINT`.
- At the end of the third wave, PP displays the message "End of program" and terminates.

Question 3 - Signal-based barrier

The program below creates two child processes. All three processes carry out `calc1()` and then `calc2()` concurrently.

```
void calc1 () {
    int i;
    for (i = 0; i < 1E8; i ++);
}

void calc2 () {
    int i;
    for (i = 0; i < 1E8; i ++);
}

int main (int argc, char * argv []) {
    int i = 0;
    pid_t pid_child [2];

    while ((i < 2) && ((pid_child [i] = fork ()) != 0))
        i ++;

    calc1 ();
    calc2 ();
    printf ( "End Process %d \n", i);
}
```

```

    EXIT_SUCCESS return;
}

```

We want to change the program by using signals (`SIGUSR1` and / or `SIGUSR2`) to make sure that no process begins `calc2` before the other two processes have also finished `calc1`.

Write the program `barrier` that implements this new specification.

Question 3.1: What is the minimum number of signal *emissions* required for this barrier?

Question 3.2: Is it possible to implement this barrier with `SIGUSR1` only?

Question 4 - Waiting for everyone without `wait()`

Consider the following program:

```

1:      #define N 4
2:      int i,j;
3:
4:      int main (int arg, char * argv []){
5:          pid_t pid;
6:          j=0;
7:          for (i=0; i<N && (pid=fork())!=0; i++)
8:              printf ("i:%d j:%d \n", i,j);
9:
10:         if (((i%2)==1) && (pid==0)) {
11:             while ((j<i) && ((pid=fork())==0)) {
12:                 j++;
13:                 printf ("i:%d j:%d \n", i,j);
14:             }
15:         }
16:     }

```





Modify this program so that the initial parent process will not terminate before any of its descendants has notified it that it is going to terminate too. Your code shall respect the following guidelines:

- Synchronization must be based on **explicit** signals.
- A process cannot perform any `wait` call.
- Your code should always promote maximum concurrency.

Submission

Assignment Text

This assignment allows submissions using both the text box below and attached documents. Type your submission in the box below and/or use the Browse button or the "select files" button to include other documents. **Save frequently while working.**



Preview

Format

Font

Size

Source

?

Words: 0

Ctrl+Right-Click to access the editor's context menu

Attachments

No attachments yet

Select a file from computer

Choose File

No file chosen

or select files from workspace or site

Submit

Preview

Save Draft

Cancel

Don't forget to save or submit!