CSE344 – System Programming
HW#2 – Report

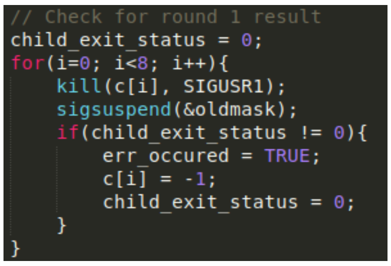
Berk Pekgöz
171044041

Solving The Problem & Design:

To solve our problem, I need to solve three synchronization problem. These three synchronizations are:

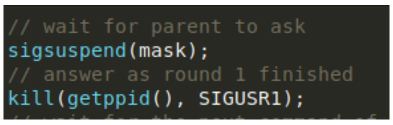
- Standard synchronization (Everything working fine).
- Synchronization when error occured.
- File content synchronization among processes.

Standard synchronization: To accomplish this synchronization I used 2-way communication. Which means, parent sends signal to child and child send feedback signal to inform parent (Mr. Aptoula said it is a good solution). For example, after all children created with fork, each child process executed and after round one finish child waits signal with sigsuspend from parent. Which means parent asks "Did you finish round one?" and then child send signal to parent to answer as "Yes i did".



```
// Check for round 1 result
child_exit_status = 0;
for(i=0; i<8; i++){
    kill(c[i], SIGUSR1);
    sigsuspend(&oldmask);
    if(child_exit_status != 0){
        err_occured = TRUE;
        c[i] = -1;
        child_exit_status = 0;
    }
}
```

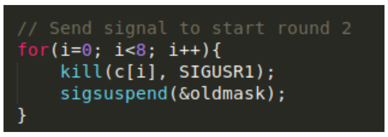
Parent: "Did you finish round one?"



```
// wait for parent to ask
sigsuspend(mask);
// answer as round 1 finished
kill(getppid(), SIGUSR1);
```

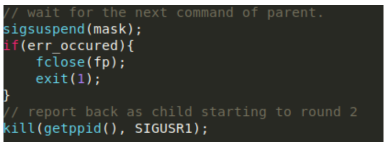
Child: "Yes, i did."

Then, parent calculates average error of round one and prints to screen. After that, parents sends SIGUSR1 signal as "Start to round two" then each child catch that signal with sigsupend and send feedback signal to inform parent as "I am starting round two".



```
// Send signal to start round 2
for(i=0; i<8; i++){
    kill(c[i], SIGUSR1);
    sigsuspend(&oldmask);
}
```

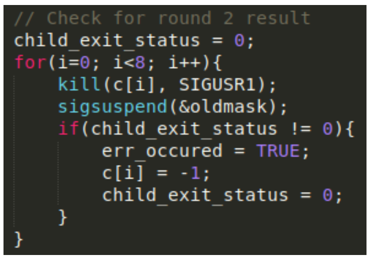
Parent: "Start to round two."



```
// wait for the next command of parent.
sigsuspend(mask);
if(err_occured){
    fclose(fp);
    exit(1);
}
// report back as child starting to round 2
kill(getppid(), SIGUSR1);
```

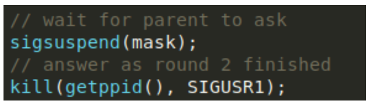
Child: "I am starting round two."

Then children executes round two. After a child finishes round two, waits signal from parent. Parent sends signal and wait signal from child one by one. This communication is “Did you finish round two?” and “Yes, i did.”.



```
// Check for round 2 result
child_exit_status = 0;
for(i=0; i<8; i++){
    kill(c[i], SIGUSR1);
    sigsuspend(&oldmask);
    if(child_exit_status != 0){
        err_occured = TRUE;
        c[i] = -1;
        child_exit_status = 0;
    }
}
```

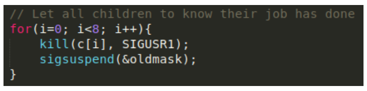
Parent: “Did you finish round two?”



```
// wait for parent to ask
sigsuspend(mask);
// answer as round 2 finished
kill(getppid(), SIGUSR1);
```

Child: “Yes, i did.”

After this parent send signal to each process to inform them as “You can terminate.”, then child catch that signal and terminates. After that parent catch SIGCHLD with sigsuspend and clean after each process.

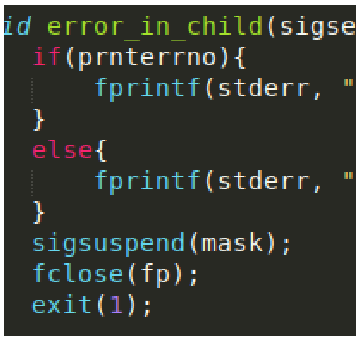


```
// Let all children to know their job has done
for(i=0; i<8; i++){
    kill(c[i], SIGUSR1);
    sigsuspend(&oldmask);
}
```

Parent: “You can terminate.”

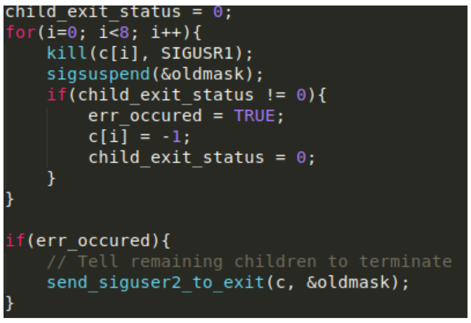
Finally parent calculates average error of round two and terminates.

Synchronization when error occured: If any error occurs in child process. Child wait for parent to ask and after that exits instead of sending SIGUSR1. With this way parent catches SIGCHLD and it is a unexpected termination so parent understand that there is a error. After that parent sends SIGUSR2 to other children to inform them to terminate.



```
id error_in_child(sigse
  if(prnterrno){
      fprintf(stderr, "
  }
  else{
      fprintf(stderr, "
  }
  sigsuspend(mask);
  fclose(fp);
  exit(1);
```

Child: Wait for parent to ask then exit.



```
child_exit_status = 0;
for(i=0; i<8; i++){
    kill(c[i], SIGUSR1);
    sigsuspend(&oldmask);
    if(child_exit_status != 0){
        err_occured = TRUE;
        c[i] = -1;
        child_exit_status = 0;
    }
}

if(err_occured){
    // Tell remaining children to terminate
    send_siguser2_to_exit(c, &oldmask);
}
```

Parent: Understands that error occured. Then terminates other children and clean after them.

File content synchronization among processes: To achive this synchronization I used file locking. To locking files i used fcntl(). With this way, a process acquire a lock and do his job. If another process try to acquire the lock of the same file then fcntl() waits that process until other process gives lock back. So, a file cannot acces the file while other file writes to that file. When second process work on the file, changes of first process will be on that file.

Reading Rows and Calculations

To read, each process locks file first then reads line by line to row_buffers. Each process reads “float” values from corresponding row of itself. Sscanf() is used for reading values from string. Then lagrange algorithm used. To achive this algorithm, I found an algorithm on internet then converted it to C code. After finding the result of lagrange, result appended to corresponding row. Then all row_buffers written to file. The same process is valid for the second round.

Notices

- Input file must be in exactly same structure with given example file.
- Coefficients are printed as higher degree to lower degree.
- Program prints ordered (Process order) because in the end each child has to give feedback to parent process in order. So, it results in ordered printing. (Actual order of executing can be tested with some prints inside of processes.)
- All kinds of synchronization between process m and her children conducted using onlySIGUSR1 and SIGUSR2
- Process m clean up after all its children synchronously, by catching the SIGCHLD signals.
- In case of an arbitrary error, exit by printing to stderr a nicely formatted informative message.
- Some function calls does not have error check because they have no valid error in our processes.
- In case of CTRL-C the program (that means all 9 processes) stop execution, return all resources to the system and exit with an information message.