Göktuğ Akın
161044018

CSE344
SYSTEM PROGRAMMING

HOMEWORK 3 REPORT

Problem definition :
In this project, The goal now is to calculate C=AxB in a distributed fashion. P1 will create 4 children processes and each of them will be responsible for calculating one quarter of C. P2 will calculate C11, P3 will calculate C12, P4 will calculate C21 and P5 will calculate C22.

Solving the problem by dividing into sub-problems :

1) Dividing the problem into 4 children processes algorithm

Assume that both input matrices are 4x4 matrices like ;

1	2	3	4		1	2	3	4		a	a	b	b
1	2	3	4		1	2	3	4		a	a	b	b
1	2	3	4		1	2	3	4		c	c	d	d
1	2	3	4		1	2	3	4		c	c	d	d

Child 1 will calculate submatrix 1 [a,a,a,a]. Child 1 accepts two input matrices the calculate quarter 1 (submatrix 1) which are described by red color.
[2x4] * [4x2] multiplication gives [2x2] sub matrix (quarter 1).

Same approach can be used to calculate any quarters.

Child 4 will calculate submatrix 4 [d,d,d,d] with same algorithm. So every children accepts two input matrices to calculate submatrix. We will send our input matrices to children with pipes.

2) Interprocess communication via pipes

Processes in this project, work on the same input matrice to do calculations. Parent Process, divides the input matrices into quarters and send this input matrices to its children by pipes. Parent forks 4 childs to complete the task. But, there should be bi-directional connection since, parent sends its children two input matrices, and children sent its output to the parent after calculation. So between parent and its children, there must be 2 pipes to communicate. So, total pipe number is 8 in this case.

3) Avoiding zombie processes by overriding the SIGCHLD signal

By handling the **SIGCHLD** signal, we are sure that, every child processes will be cleared from process table immediately. But if 2 processes dies at the same time, SIGCHLD handler of parent will get 1 SIGCHLD signal, since in POSIX, signals are not queued. So, inside handler, I used while(*waitpid(-1,NULL,WNOHANG)*) system call to reap all childs.

At the end of the main, we must wait to all of our children. Because If we don't wait our children, parent process can die before its children. In this case, zombie processes reaped by ***init*** process since childs become orphans. But this is bad design, we have to reap all of our childs inside parent process.

4) Interrupts at system calls

Lets look the *linux manual page (7)* signal.

- Interruption of system calls and library functions by signal handlers. If a signal handler is invoked while a system call or library function call is blocked, then either ;
    - the call is automatically restarted after the signal handler returns.
    - the call fails with the error **EINTR**.

- If a blocked call to one of the following interfaces is interrupted by a signal handler, then the call will be automatically restarted after the signal handler returns if the **SA_RESTART** flag was used; otherwise the call will fail with the error **EINTR**:
    - read(), write() from pipes, terminal etc..

While working on a pipe, read / write operations can be interrupted by signals. In our project, if child processes send SIGCHLD to its parent while parent is at the read system call, this call be interrupted and return error EINTR default.

We can overcome this problem by three methods ;

- Blocking the all signals before read operation. So read operation on pipes accepted as critical section, can not interrupted by any signal.
- Add SA_RESTART flag to signal handler of SIGCHLD signal. The kernel will automatically restart the system call which is interrupted by SIGCHLD signal.(deprecated in the project)
- Add manual restart macro. Since not all of system calls supports the SA_RESTART flag, it is good to write our restart macro.

I used third method to overcome this problem by following macro definition ;

- #define NO_EINTR(stmt) while ((stmt) == -1 && errno == EINTR);