Gebze Technical University

Department Of Computer Engineering

CSE 312 /CSE 504 Spring 2023

Operating Systems

Homework 01

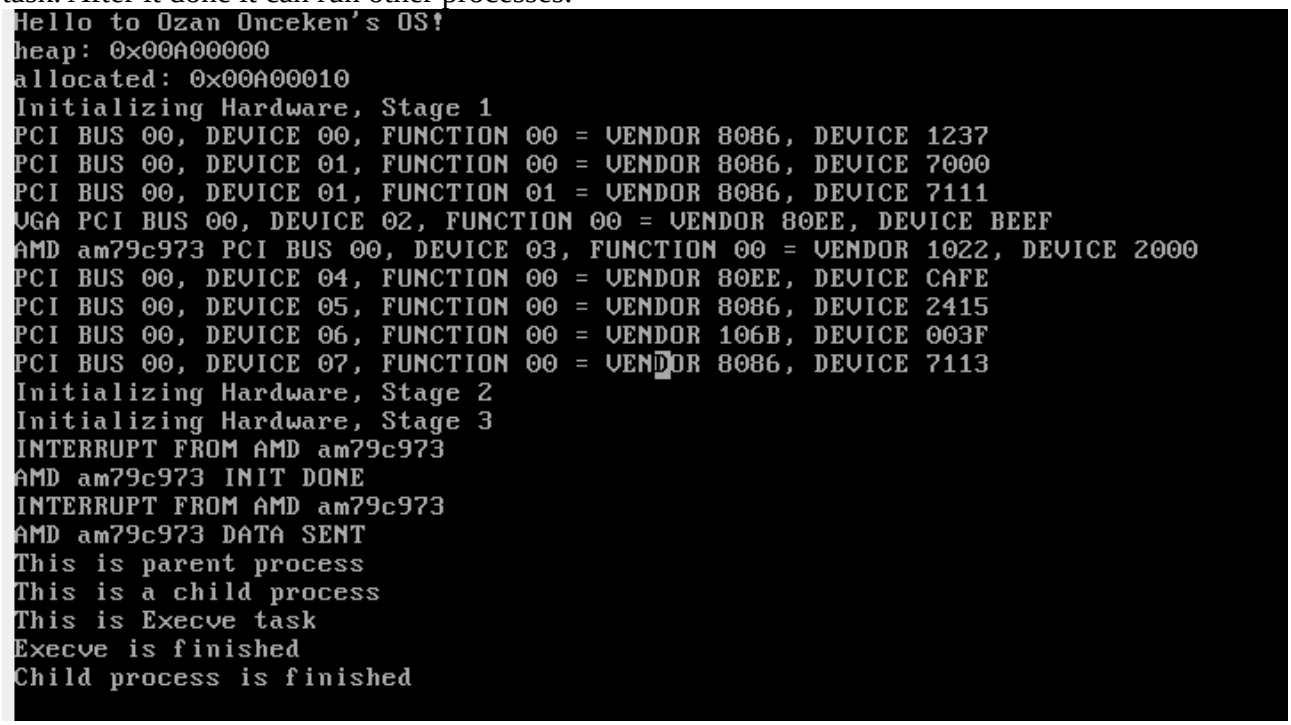
10.04.2023

Ozan Argıt ÖNÇEKEN

1901042259

execveHelper

This is a helper test function for execve. When the execve command starts it only runs that task. After it done it can run other processes.



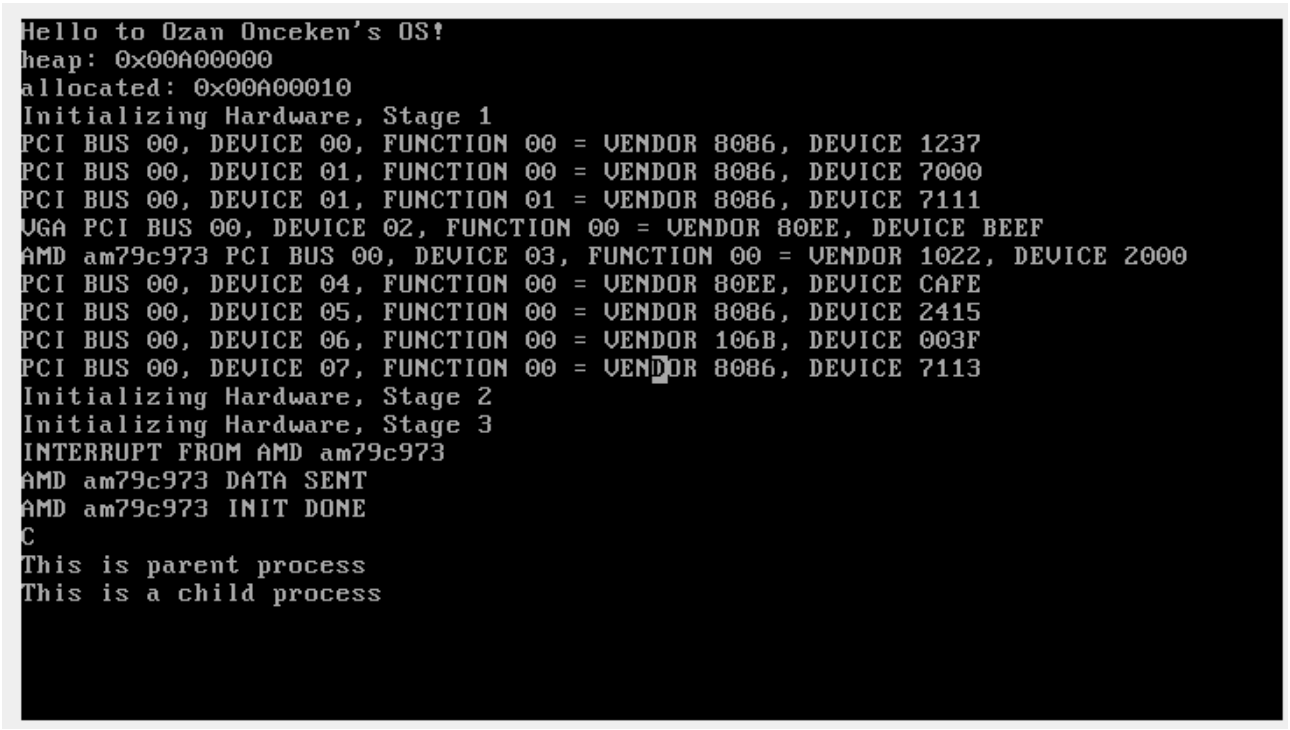
ForkWait

This task is for testing the fork, wait and execve.

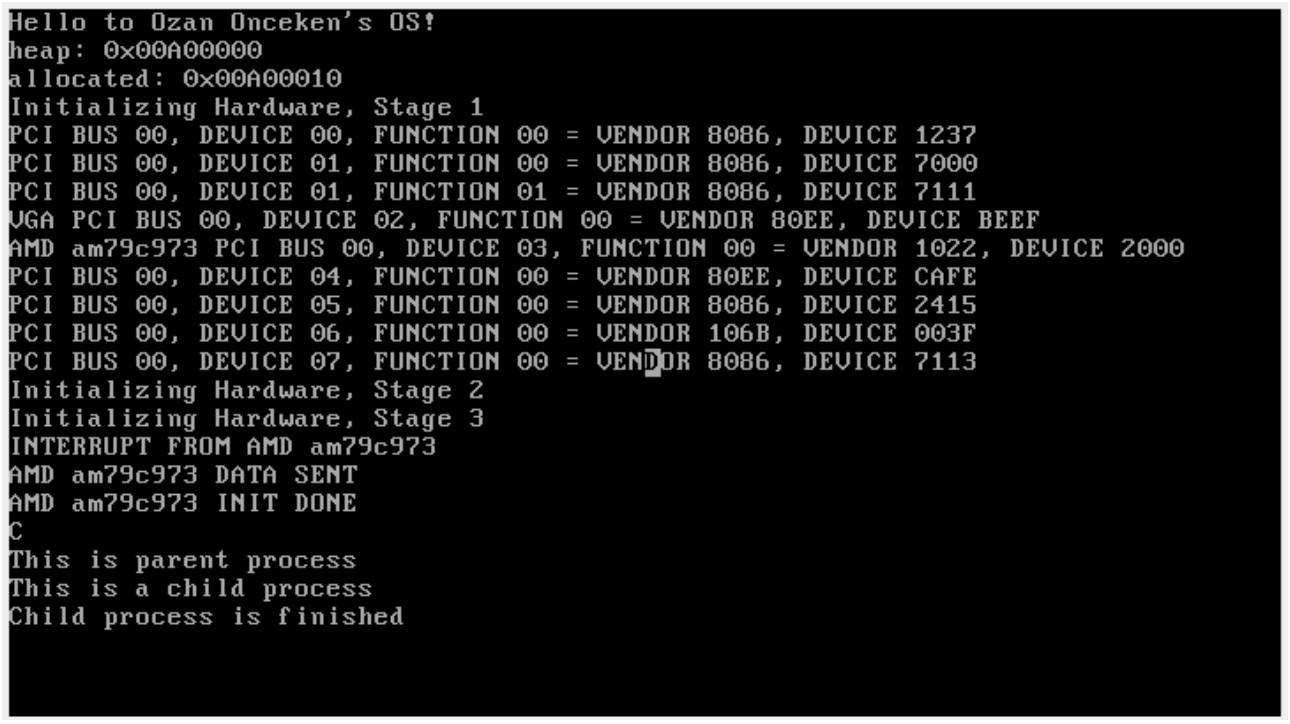
At first ForkWait forks itself and it holds the returned value in "x" variable. If it is parent process it prints "This is parent process". If it returns another think it prints "This is a child process".

The parent process waits the child with my_sleep function. my_sleep function is not best sleep function but it is nice enough to test waitpid. my_sleep function is basically: for(int i=0;i<999999;i++) and it makes it seem like waiting while the other process (it can be done sometimes before the child process worked) is running. The child process locks its parent while it is running.

In summary parent process prints "This is parent process" waits and prints "Child process is done" . Child process prints "This is a child process" and waitpid().



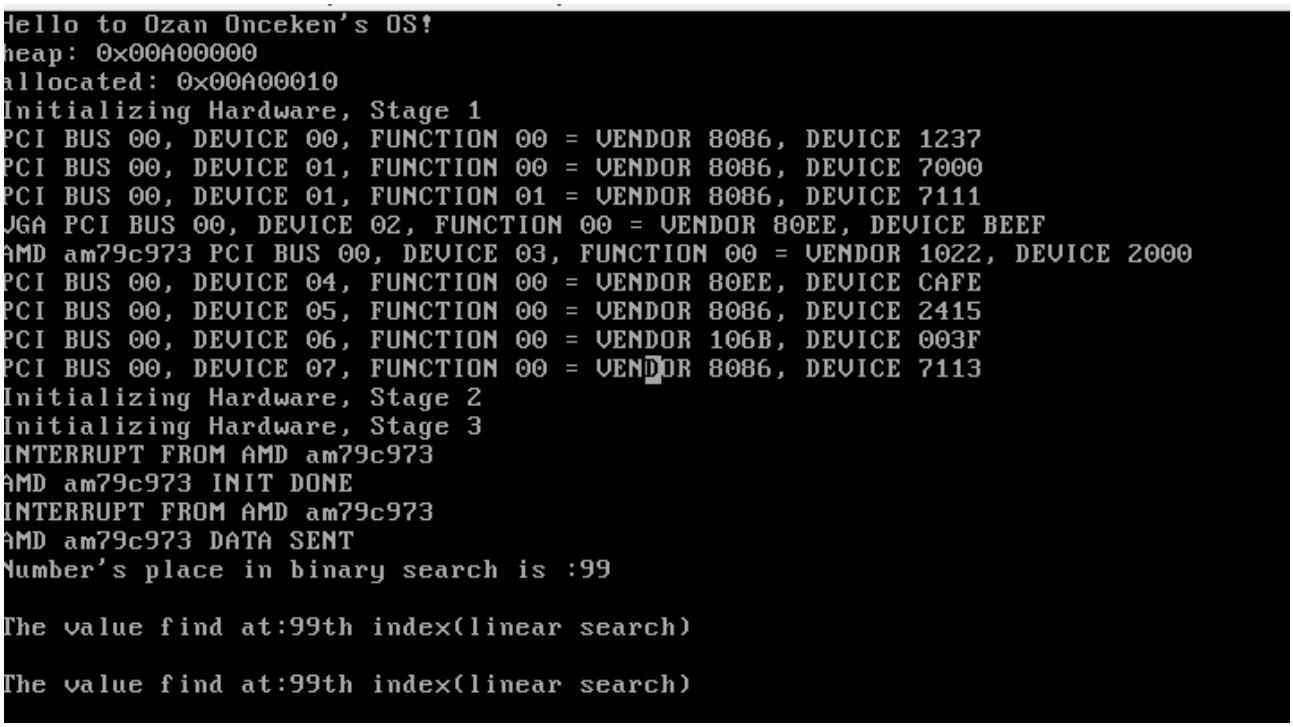
Parent process worked and waiting for child process.



When child process finished it lets the parent process continue.

First Strategy

At first strategy I am forking the process and printing linear search and binary search.



First strategy forks 2 times but fork returns 0 and the other processes id and I can not take the third id and second id's difference. So system writes the binary search's output and linear search's output but not collatz.

Second Strategy

The second strategy runs 10 fork in a for loop and executes linear search 10 times.

The 8 of them are fitting in screen so I put 8 output's screenshot.

Final Strategy

Final strategy uses addtask method only.

Fork:

The algorithm of fork is copy the current process and add put it in Tasks array. Fork method arranges the parent and child id's of processes. The bad part of fork is if a program forked 2 times there should be 4 process but in this system there are 3. For this fork I made a new Task creator that takes the process and sets variables to new task. The cpustate copying is important at that point.

Waitpid:

waitpid method takes the process id and finds that processes parent and set it wait for child (with some boolean variables) and sets the child processes boolean variable is runned after wait to false. When Scheduler comes to run parent it will not run it because of the waiting variable. When it came the children it runs the children and after that it finds the parent process and sets it runnable and everything same again so there will be no problem.

Execve:

Execve takes the id of the process and makes every process blocked except the given process. When the given process runs Scheduler makes every other process runnable again. So that the execve makes desired process run and others wait.

KillId:

KillId method takes the id of the process and finds it in the Tasks array and makes it taskState variable 0 so scheduler will not run it.

Kill:

Kill method takes the current process and makes it's taskState 0 so Scheduler do not run it.

Changes in Scheduler:

If task state is 1 it can run the task. So I created a getStateOfTask() getter method that returns the task's taskState. If it is not 1 scheduler does not runs the task. After that if task is waiting for execve or waiting for child scheduler will not run it either.

If it passes from that controls and it is a child process that parent are waiting for it it will make that tasks is runned after wait is true so the parent process can run now so Scheduler finds the parent of it and sets the parent processes isWaiting variable to false and when the parent process takes the order it can run that time. In summary when child runs Scheduler sets parent runnable.

If Scheduler finds out that process is wanted from execve, sets its isExeced variable to false it means next time Scheduler does not do the same thing and sets all other processes isWaitingForExecve value to false so next time they will not wait. After that Scheduler returns the task which execed before.

If the Scheduler can not return that current process it searchs any available process to return for not take any interrupt.

General Rules About Project

I have added screenshots taking the other processes behind comments because if I did not do it processes did not fits in the screen if user wants to see them separately he should use take some tasks to comment lines. (starting with comment MAIN and ends with END OF MAIN lines 416-438 in kernel.cpp)

I wish I make the process killing with system calls but I did not enough time. So I made all process kill itself with usage of taskManager.kill(). This method makes task unavailable.

There can be unnecessary variables because of I used them and forget to delete them.

Functions: Functions are easy for test they are creating their own arrays and does their own jobs.

Binary search and linear search creates an array includes number from 0 to 99. After that they searches their element 99.

Collatz creates the array too and apply the collatz test.

Binary search ,linear search and collatz prints their own prints when they finished their processes.

PLEASE give information to me if my code does not compiles or works on your computer. I can make a demo from my computer.