Explanation Part-2

In general: What does the OS do when it starts?

One of the first things the OS does is creating the first process. After that the first process creates other processes and each process can create new processes as well. The Linux system call to create a new process is the fork statement.

OS saves the information for each process (such as its ID, its parent ID PPID,) in a structure called Process Control Block (PCB). The project defines what needs to be in the PCB.

Another thing that the OS does is to find out whose turn it is to use the CPU. The scheduler has an algorithm for deciding that. Processes wait in a queue until it is their turn to use the CPU. Once the time for a process using the CPU is up, the OS saves all the register values in the PCB entry for that process. Also OS will transfer the saved register values of the new process from its PCB entry to the CPU structure registers.

In the next phase of the project, complete the child process code (Process Manager).

The process manager process simulates four process management functions: **creation of new (simulated) processes**, **replacing the current process image of a simulated process with a new process image**, **management of process state transitions**, and **process scheduling**. In addition, it spawns a reporter process whenever it needs to print out the state of the system.

We need to define the following structures in our code:

2.1 Data structures:

The process manager maintains six data structures: *Time, Cpu, PcbTable, ReadyState, BlockedState, and RunningState*.

1. *Time* is an integer variable initialized to zero.
2. *Cpu* is used to simulate the execution of a simulated process that is in running state. It should include data members to store a pointer to the program array, current program counter value, integer value, and time slice of that simulated process. In addition, it should store the number of time units used so far in the current time slice.
3. *PcbTable* is an array with one entry for every simulated process that hasn't finished its execution yet. Each entry should include data members to store process id, parent process id, a pointer to program counter value (initially 0), integer value, priority, state, start time, and CPU time used so far.
4. *ReadyState* stores all simulated processes (PcbTable indices) that are ready to run. This can be implemented using a queue or priority queue data structure.
5. *BlockedState* stores all processes (PcbTable indices) that are currently blocked. This can be implemented using a queue data structure.
6. *RunningState* stores the PcbTable index of the currently running simulated process.

This is an example of pcbEntry:

```
class PcbEntry {

public:

    int processId;

    int parentProcessId;

    vector<Instruction> program;

    unsigned int programCounter;

    int value;

    unsigned int priority;

    State state;

    unsigned int startTime;

    unsigned int timeUsed;

};
```

Create several files and write in it some simple programs such as the following program defined in the project. Note that the instruction set of our CPU is simple and has operations such as S, A, D, F or R. Become familiar with the meaning of each instruction.

```
S 1000
A 19
A 20
D 53
A 55
F 1
R file_a
F 1
R file_b
F 1
R file_c
```

```
F 1
R file_d
F 1
R file_e
E
```

The CPU must fetch each instruction and execute each instruction. It should also update the PC (Program Counter) to point to the next instruction.