SYSC4001 Assignment 2 – Part III report

Objective

This assignment is based on the interrupt simulator of Assignment 1, and extends the behavior of fork() and exec() system calls. The simulator reads the input trace file, updates the process control block table according to commands, allocates fixed memory partitions, and records system execution logs to analyze the process of process creation and program loading.

Simulator Components

Each system call triggers an interrupt: switch kernel state -> save context -> search for interrupt vector -> execute ISR -> IRET return. These steps are all in the file named execution.txt. The process control block will save information such as PID, program name, memory partition number, and size for each process. After each occurrence of FORK or EXEC, the system will print the current PCB table in the file named system_status.txt. Memory partition management will allocate memory into 6 partitions, namely 40, 25, 15, 10, 8, and 2 MB. It adopts a first adaptation allocation algorithm from small to large; When using exec(), release the old partition and reassign a new partition. The scheduling strategy will prioritize the running of child processes while the parent process enters a waiting state. The simulator is a single CPU, non preemptive execution model.

Simulation Results

Test 0 – Basic Fork + Exec

The init process executes fork, the child process runs program1, and the parent process runs program2. The child process executes first, and the parent process waits. The PCB table displays the running/waiting status and partition number correctly. The output is consistent with the reference results provided by the professor.

Test 1 – Nested Fork inside Exec

The child process runs program1, and it forks internally again. The two child processes respectively execute program2. The execution order is child process -> descendant process -> parent process. Recursively calling simulate_trace() correctly handles multi-layer fork/exec.

Test 2 – Exec with I/O

The child process executes a CPU burst, while the parent process runs program1. The log displays the complete interruption process. The device delay comes from Device_table.txt, and the time overlay is correct.

Test 3 – Multi-level Exec

The child process runs program6, program7, and program8 in sequence, while the parent process runs program7. The process image has been replaced multiple times and is running normally. Re partitioning and loading time meet expectations

Test 4 – Double Fork

The init process forks twice, with two child processes running program9 and the parent process running program10. There are three processes in the system that are independent of each other. The same program can still allocate partitions correctly when loaded by multiple processes.

Discussion

The function of fork() is to copy the parent process PCB, allocate new partitions, wait for the parent process to enter, and execute the child process first. The function of exec() is to release old partitions, reallocate and load external programs. The time difference in the log reflects the impact of program size on loading latency. The PCB table accurately reflects each state switch and memory usage. The recursive logic of the simulator accurately simulates the creation and execution of multi-layer processes in real systems.

Conclusion

All five tests were successfully executed. The simulator accurately reproduced the behavior of fork() and exec() on a single CPU and fixed memory partition. The execution log and system state table are consistent with expectations, fully reflecting core mechanisms such as process creation, program loading, and interrupt handling.