xv6-public

Enhancing the fate of this noob os

So the few changes made yet are:-

1. Adding 3 system calls(waitx, ps, set_priority)
2. Adding 2 user calls(setPriority, setPriority)
3. Adding 3 more scheduling algorithms along with the default one

Now let's understand how I have made the changes.

System Calls

(note that changes have to be made in proc.c, user.h, user.S, syscall.c, syscall.h and sysproc.c for each syscall)

1. waitx(int* wtime, int* rtime)

Logic:

- The main code resides in proc.c where we just check for the first zombie childs of the calling parent, calculate the wait time by the formula

  wtime = p -> etime - p -> ctime - p -> rtime
- We return the pid of the child we waited for just like wait command.

Implementation: Busy waiting until a child becomes zombie and then do the calculation and other important arrangements.

Return type: int(pid of child)

2. pls() (this is basically the ps command but with a different name)

logic:

- Just check the ptable.proc for process that have existed and print out the usable information.
- Note that there are 2 cases for calculating the wait time of a process due to the possibility of a process current dead or not so.

Implementation: Just a basic for loop and printing information of processes with a pid above zero.

Return type: int(could be void as well but I did for my convenience)

3. set_priority(int pid, int priority)

logic:

- Find the process with the given pid and change its priority to the said value.
- Also return old priority for reference.

Implementation: Simple loop to check for the process with the given id(if it is still existent)

Return type: int(previously held priority for the process)

Usercalls (like the commands in bash/zsh)

(Dont have to do much, just see if required system calls exist. Next, make a c program and make apt changes in makefile to run this c file and get the experience of a user call)

- No specific telling required here.
- setPrioirity: Called by c file named "setPriority.c"
- ps: Called by c file named "ps.c"

Scheduler Enhancement

(added FCFS, PBS and MLFQ scheduling algorithms alongwith the default RR algorithm)

FCFS(First Come First Served)

- The obvious happens here. Each process is given a full processing power until it is done with its processing
- The basis of prioritization is the time when each process was created.
- Note that this algorithm is not starvation proof.
- Major changes occur inside the scheduler function.

PBS(Priority Based Scheduling)

- Each process is assigned a priority of 60 by default and this priority can be changed by the use of set_priority system call or setPriority usercall.
- To implement this method, we look at ptable for the process with the lowest value of priority(viz maximum preference)
- If we find process with same maximum priority(preference), we must run the RR algorithm.
- The priorities of the processes can be changed by using system call or user call for them use.
- Major changes in the scheduler function along with other necessities for system call.

MLFQ(Multi-Level Feedback Queue)

- By definition, an MLFQ algorithm is supposed to solve starvation and I/O wait problems.
- We have add_proc_to_q and remove_proc_from_q functions to move processes between the queues as per our requirement
- 5 queues with varying priorities and 1, 2, 4, 8 & 16 ticks resp. have been made.
- Ageing has been made possible by storing the age of each process and promoting it once we see it has grown over-age.
- We basically go looking at each of the processes in each queue and alter the information for each as we give each one their go at processing.

Short Report on performance of Scheduling algorithms

- For a test user program named benchmark(available in the repo) with no. of processes equal to 5, the ticks required to complete the execution was noted.

1. RR 1168
2. FCFS 2893
3. PBS 1293
4. MLFQ 3225

- RR and PBS seem to perfrom better than others due to their minimal complexity, although RR seems to win the battle.
- FCFS is a very primitive algorithm therefor its bad performance is justified.
- Surprisingly, MLFQ performs the worst possibly because the no. of processes being tested is not so close to the real world simulation.

I don't think this os is noob anymore!