The design

FIFO:

The scheduler runs on core 0 and all children run on core 1. As soon as a new job is ready, the scheduler forks the job and set the priority of the child using `sched_setscheduler()`. I use linux's "SCHED_FIFO" real-time scheduler and set the children's priorities to do FIFO scheduling.

RR:

The scheduler (main process) runs on core 0 and children (jobs) run on core 1. As soon as a new job is ready, the main process forks it and set it to lowest priority (SCHED_IDLE) and also push the job into a "readyQueue". At the same time main process keeps track of the status of children in the readyQueue with the `waitpid()` system call. If a child has ended, it is removed from the readyQueue. When a child has run for 500 time quantums, the child is put back to the end of readyQueue and another child is started.

SJF:

In this case, there are two priority queues, createQueue and readyQueue, which are sorted according to the ready time and the execution time of jobs respectively.
Initially, all jobs are in createQueue. In each loop, the scheduler will take out all the ready jobs, fork them, idle them, and push them into readyQueue. If no job is running, the scheduler will take out one job from readyQueue and wake it up. After that, the scheduler will idle all the waiting jobs.

PSJF:

My design is to have a scheduler mainly executed on CPU0 (with some time interval switched to CPU1), while children forked all restricted to CPU1. When CPU0 and CPU1 running concurrently, they both execute a loop with one statement which representing 1 unit time. Idealy, the timing on scheduler will be the same as that on CPU1, and then I can simulate a process arrival on the correct ready time using scheduler itself. (I use blocking wait to sync the timing on two CPU.) As for the implementation of scheduler, I use a linked list sorted by remain execution time to keep track of the priority of children forked. When a new process arriving, I first locate it the correct position in list, and then assign it a priority value according to its neighborhoods' priorities. As for the forking issue, I switch the scheduler from CPU0 to CPU1 and assign it the highest priority when scheduling a new arriving process (not simulating the timing), for theoretically, the CPU control (on single processor system) is transferred to scheduler in this time interval, leaving all children processes suspended. After scheduling, I fork the new process while scheduler is still on CPU1, and then the new process inherits the CPU and priority settings. Finally, I switch the scheduler back to CPU0, leaving new process in CPU1 with the highest priority, and then the new process set itself the priority value given by the scheduler. In this way, the new process can first initialize some issues (for instance, starting time) before setting its priority and truly scheduled by the kernel scheduler.

The results

FIFO:

FIFO_1.txt

Terminal output:

P1 15611

P2 15612

P3 15613

P4 15614

P5 15615

Kernel dmesg:

[Project1] 15611 1461344484.922188044 1461344485.751088858

[Project1] 15612 1461344484.922270060 1461344486.582002401

[Project1] 15613 1461344484.922333479 1461344487.418671608

[Project1] 15614 1461344484.922413588 1461344488.251573801

[Project1] 15615 1461344484.922473907 1461344489.087369442

FIFO_2.txt

Terminal output:

P1 26663

P2 26674

P3 26675

P4 26686

Kernel dmesg:

[Project1] 26663 1461344704.446754932 1461344839.375505447

[Project1] 26674 1461344704.618067026 1461344847.772628784

[Project1] 26675 1461344704.818082333 1461344849.514639616

[Project1] 26686 1461344704.923933268 1461344851.190951824

FIFO_3.txt

Terminal output:

P1 2412

P2 2433

P3 2444

P4 2455

P5 2456

P6 2457

P6 2468

Kernel dmesg:

[Project1] 2412 1461344861.660900116 1461344875.189916849

[Project1] 2433 1461344862.042437315 1461344883.571819782

[Project1] 2444 1461344862.178107738 1461344888.689447641

[Project1] 2455 1461344862.344671011 1461344890.421060562

[Project1] 2456 1461344862.544667721 1461344892.164105177

[Project1] 2457 1461344862.544744492 1461344893.902858257

[Project1] 2468 1461344862.674362421 1461344900.768971443

RR:

RR_1.txt

Terminal output:

P1 15837

P5 15841
P4 15840
P2 15838
P3 15839
Kernel dmesg:
[Project1] 15839 1461486239.987489462 1461486243.234455824
[Project1] 15837 1461486239.987215757 1461486244.068005562
[Project1] 15840 1461486239.987373114 1461486244.070775270
[Project1] 15838 1461486239.987437487 1461486244.074333191
[Project1] 15841 1461486239.987305641 1461486244.082593918

RR_2.txt

Terminal output:
P1 19778
P2 19799
Kernel dmesg:
[Project1] 19778 1461486319.649528503 1461486332.719804525
[Project1] 19799 1461486319.935198784 1461486334.373989820

RR_3.txt

Terminal output:
P1 23318
P2 23406
P3 23499
P4 23600
P5 23631
P6 23682
Kernel dmesg:
[Project1] 23499 1461486394.531195402 1461486422.454628229
[Project1] 23318 1461486390.635202885 1461486424.154150486
[Project1] 23406 1461486392.575202465 1461486424.173103571
[Project1] 23682 1461486398.175189495 1461486436.440509319
[Project1] 23631 1461486397.175201178 1461486438.906811953
[Project1] 23600 1461486396.507192373 1461486439.738892317

SJF:

SJF_1.txt

Terminal output:
P2 2957
P1 2956
P3 2958
P4 2959
Kernel dmesg:
[Project1] 2957 1461403041.064632177 1461403061.031341314
[Project1] 2958 1461403042.753066063 1461403073.633394718
[Project1] 2959 1461403044.439320803 1461403113.535823822
[Project1] 2956 1461403041.064227581 1461403172.673226595

SJF_2.txt

Terminal output:

P2 2962
P4 2963
P3 2964
P1 2965
P5 2966
Kernel dmesg:
[Project1] 2962 1461403221.624680519 1461403222.617942572
[Project1] 2966 1461403223.112521410 1461403225.511780977
[Project1] 2963 1461403221.625031471 1461403261.639800787
[Project1] 2964 1461403223.112000704 1461403303.865467072
[Project1] 2965 1461403223.112318277 1461403372.425801992

SJF_3.txt

Terminal output:
P1 2969
P3 2970
P2 2971
P4 2973
P5 2972
P6 2974
P7 2975
P8 2976
Kernel dmesg:
[Project1] 2969 1461403439.304888964 1461403463.117170572
[Project1] 2973 1461403441.140427351 1461403463.816141367
[Project1] 2972 1461403441.140084743 1461403463.826304913
[Project1] 2974 1461403442.741104841 1461403507.783749580
[Project1] 2975 1461403444.464551210 1461403535.218012094
[Project1] 2971 1461403439.305577517 1461403556.303640127
[Project1] 2970 1461403439.305319071 1461403587.281930923
[Project1] 2976 1461403445.760657072 1461403618.707913160

PSJF:

PSJF_1.txt:

Terminal output:
P1 1923
P2 1924
P3 1925
P4 1926
Kernel dmesg:
[Project1] 1926 1461507244.997638702 1461507250.771345139
[Project1] 1925 1461507242.990591288 1461507258.580595732
[Project1] 1924 1461507240.894872904 1461507269.958167791
[Project1] 1923 1461507238.916940689 1461507287.469169855

PSJF_2.txt:

Terminal output:
P1 1975
P2 1977

P3 1978
P4 1979
P5 1980
Kernel dmesg:
[Project1] 1977 1461508725.691853046 1461508727.725966215
[Project1] 1975 1461508723.630764484 1461508731.951124668
[Project1] 1979 1461508733.853814125 1461508737.955019712
[Project1] 1980 1461508737.958183050 1461508739.857605219
[Project1] 1978 1461508727.730339289 1461508745.633692741

PSJF_3.txt:
Terminal output:
P1 2250
P2 2251
P3 2252
P4 2253
Kernel dmesg:
[Project1] 2251 1461509247.498799086 1461509248.525429010
[Project1] 2252 1461509248.531804323 1461509249.545227289
[Project1] 2253 1461509249.556759119 1461509250.532097340
[Project1] 2250 1461509246.542829275 1461509253.498157263

The comparisons and explanation

FIFO:
My results actually fit well with the theoretical results. This may due to linux's well-crafted SCHED_FIFO scheduler.
RR:
The results fit well with the theoretical results. Explanations are already mentioned in the "The design" part of this report and comments in the source code.
SJF:
The result is close to theoretical value. To minimize the time difference between scheduler and child processes, the scheduler waits for the running job to finish.
Also, the scheduler has to set all the waiting processes to IDLE in every loop, or they may wake up by themselves, so the performance may not be good.
PSJF:
The results also fit well with the theoretical results. In my machine, the time measurement is about: **1 second / 500 unit time**, fitting well with the experiment results pasted above.

The contributions of each member

Kernel: b03902082 江懿友
FIFO: b03902082 江懿友
RR: b03902082 江懿友
SJF: b03902126 高翊軒
PSJF: b03902130 楊書文