OS 2020 project

– Project 1 –

B06902019 資工三 洪佳生

Kernel

- **Version**:
  Linux 4.14.25 -> https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.14.25.tar.xz
- **Added syscall 333: sys_my_clock()**
  - Get the data by getnstimeofday()
- **Added syscall 334: sys_my_printk()**
  - Prints a string to dmesg

Design

main.c

- Read the input (number of process, scheduling policy, process name, ready time, execution time)
- Use scheduler() funtion to implement sheduling.

process.c

Define the followin function:

- TIME_UNIT(): define a basic unit of execution time.

- assign_proc_core(pid, core): decide which core the process "pid" will run on.

- proc_out(pid): use sched_setscheduler() to reduce the priority "pid" of process and then assign the child process back to the core 0 where the parent process is running.

- proc_wakeup(pid): assign the child process to the core where the core 1 and then use sched_setscheduler() to raise the priority "pid" of process.

- proc_exec(Process):

  (1) Scheduling process call fork() to simulate the process which is ready and stops child process by reducing its priority until parent process wakes it up.Because parent and child process are in the same core. So due to the priority,child process won't run if it shouldn't run.But in case the child process run in the unavailable time and unfortunately start the timer, we set a while() loop to avoid the above problem and it will break the loop when child process priority is raised by parent process.

  (2) When the timer starts, child process will enter a while() loop for execution time of TIME_UNIT().

  (3) When the timer ends, use system call to output the message into dmesg.

scheduler.c

- First, we will assign a particular core 0 to scheduling process and raise its priority to the highest level to prevent potential preemptive problem between scheduling process and the child processes which are are generated by fork().

- Second,initializes child process by -1 to represent not ready process or already finished process.

- In while(1) loop, we will kepp doing tje following five steps until all processes are done.

  Step1:
  Check whether there are some process are already done in last UNIT_TIME.If so,label process's pid into -1 and finished processes number += 1.If finished processes number is equal to total process number,break the while(1) loop and finish scheduling.

  Step2:
  Check whether there are some processes which are ready and if so,implement proc_exec().

  Step3:
  Use switch to choose the scheduling policy to find the next process to implement it.

  There are four policies:
  Assume there are some ready processes.
  FIFO():
  When the implementing process i finish,the next implement process will be i+1.
  SJF():
  When the implementing process i finish,the next implement process will be the shortest execution time process in the ready queue.
  RR():Implementing in queue.When process is ready,it will be pushed into the queue. In each time slice 500 UNIT_TIME or the running process **i** finish in time slice,the next running process will be select by the header of ready queue and then if process **i** is not finished yet,push it back to the queue.
  PSJF():
  NO matter the implementing process i is finished,the next implement process will be the shortest execution time process in the ready queue.

  Step4:
  If the next running process isn't the same as now running process,then scheduling process will reduce the priority of now runnig process and raise the priority of next running process.
  Step5:
  run a TIME_UNIT in parent and now runnig child process simutaneously.

Result

Error Rate: round off to the 4nd decimal place

Start time, End time: theoretical time (calculate by math)

Expect exec time:the time the process should finish

My start time, My end time:the time I get from my program (Initialize by minimun My start time)

My exec time:the time the process finish in my program

Error rate: the error between Expect exec time & My exec time

UNIT_TIME

unit time	**0.001387**

FIFO_1.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P1	0	500	500	0.00	516.51	516.51	3.30%
P2	500	1000	500	516.63	1004.95	488.31	2.34%
P3	1000	1500	500	1005.06	1518.15	513.09	2.62%
P4	1500	2000	500	1518.36	2019.46	501.09	0.22%
P5	2000	2500	500	2019.58	2521.72	502.14	0.43%

FIFO_2.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P1	0	80000	80000	0.00	79668.77	79668.77	0.41%
P2	80000	85000	5000	79668.92	84594.43	4925.51	1.49%
P3	85000	86000	1000	84594.59	85599.33	1004.74	0.47%
P4	86000	87000	1000	85599.47	86596.21	996.74	0.33%

FIFO_3.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P1	0	8000	8000	0.00	7959.73	7959.73	0.50%
P2	8000	13000	5000	7959.85	12852.55	4892.70	2.15%
P3	13000	16000	3000	12852.67	15722.23	2869.56	4.35%
P4	16000	17000	1000	15744.11	16738.42	994.31	0.57%
P5	17000	18000	1000	16738.57	17715.99	977.42	2.26%
P6	18000	19000	1000	17716.11	18709.06	992.95	0.71%
P7	19000	23000	4000	18709.17	22574.96	3865.79	3.36%

FIFO_4.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P1	0	2000	2000	0.00	1975.96	1975.96	1.20%
P2	2000	2500	500	1976.08	2460.45	484.37	3.13%
P3	2500	2700	200	2460.57	2651.03	190.46	4.77%
P4	2700	3200	500	2657.05	3148.83	491.78	1.64%

FIFO_5.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P1	0	8000	8000	0.00	7599.11	7599.11	5.01%
P2	8000	13000	5000	7625.62	12494.19	4868.57	2.63%
P3	13000	16000	3000	12494.36	15456.67	2962.30	1.26%
P4	16000	17000	1000	15456.78	16466.10	1009.32	0.93%
P5	17000	18000	1000	16466.24	17484.94	1018.70	1.87%
P6	18000	19000	1000	17485.08	18477.26	992.18	0.78%
P7	19000	23000	4000	18477.37	22306.31	3828.94	4.28%

PSJF_1.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P4	3000	6000	3000	2926.36	5910.21	2983.86	0.54%
P3	2000	10000	8000	1938.06	9980.34	8042.28	0.53%
P2	1000	16000	15000	964.02	16151.30	15187.29	1.25%
P1	0	25000	25000	0.00	24838.03	24838.03	0.65%

PSJF_2.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P2	1000	2000	1000	972.48	1941.84	969.36	3.06%
P1	0	4000	4000	0.00	3933.70	3933.70	1.66%
P4	5000	7000	2000	4895.65	6938.06	2042.41	2.12%
P5	7000	8000	1000	6938.26	7950.06	1011.80	1.18%
P3	4000	11000	7000	3933.88	10766.87	6832.99	2.39%

PSJF_3.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P2	500	1000	500	496.07	987.82	491.75	1.65%
P3	1000	1500	500	988.04	1495.94	507.90	1.58%
P4	1500	2000	500	1496.13	1999.34	503.21	0.64%
P1	0	3500	3500	0.00	3408.00	3408.00	2.63%

PSJF_4.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P3	100	1100	1000	96.53	1091.75	995.22	0.48%
P2	0	3000	3000	0.00	2924.32	2924.32	2.52%
P4	3000	7000	4000	2924.47	6817.48	3893.00	2.67%
P1	7000	14000	7000	6817.59	13735.83	6918.24	1.17%

PSJF_5.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P1	100	200	100	0.00	97.93	97.93	2.07%
P3	200	400	200	98.17	296.47	198.30	0.85%
P2	400	4400	4000	297.08	4252.56	3955.48	1.11%
P4	4400	8400	4000	4252.67	8276.24	4023.57	0.59%
P5	8400	15400	7000	8276.35	15146.34	6869.99	1.86%

RR_1.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P1	0	500	500	0.00	502.69	502.69	0.54%
P2	500	1000	500	504.92	1004.04	499.12	0.18%
P3	1000	1500	500	1004.18	1469.74	465.57	6.89%
P4	1500	2000	500	1479.77	1950.90	471.13	5.77%
P5	2000	2500	500	1967.47	2447.09	479.62	4.08%

RR_2.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P1	600	8100	7500	0.00	7410.11	7410.11	1.20%
P2	1100	9600	8500	522.87	8809.75	8286.88	2.51%

RR_3.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P3	4200	18200	14000	2876.78	16502.61	13625.83	2.67%
P1	1200	19700	18500	0.00	18030.84	18030.84	2.54%
P2	2700	20200	17500	1444.28	18593.91	17149.63	2.00%
P6	8200	28200	20000	6686.48	26321.09	19634.61	1.83%
P5	6700	30200	23500	5248.80	28410.97	23162.17	1.44%
P4	6200	31200	25000	4785.54	29295.44	24509.89	1.96%

RR_4.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P4	1500	5500	4000	1450.92	5448.00	3997.07	0.07%
P5	2000	6000	4000	1946.22	5939.40	3993.18	0.17%
P6	2500	6500	4000	2426.85	6443.11	4016.26	0.41%
P3	1000	14500	13500	971.99	14327.21	13355.22	1.07%
P7	3500	18500	15000	3462.55	18329.50	14866.95	0.89%
P2	500	20000	19500	478.12	19790.00	19311.88	0.96%
P1	0	23000	23000	0.00	22905.97	22905.97	0.41%

RR_5.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P4	1500	5500	4000	1448.02	5393.69	3945.67	1.36%
P5	2000	6000	4000	1945.35	5902.72	3957.37	1.07%
P6	3000	7000	4000	2920.18	6900.18	3980.00	0.50%
P3	1000	14500	13500	952.83	14257.71	13304.88	1.45%
P7	3500	18500	15000	3417.34	18255.21	14837.88	1.08%
P2	500	20000	19500	475.65	19761.38	19285.74	1.10%
P1	0	23000	23000	0.00	22691.93	22691.93	1.34%

SJF_1.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P2	0	2000	2000	0.00	1990.87	1990.87	0.46%
P3	2000	3000	1000	1991.00	2977.56	986.56	1.34%
P4	3000	7000	4000	2977.67	6941.97	3964.30	0.89%
P1	7000	14000	7000	6942.08	14002.82	7060.75	0.87%

SJF_2.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P1	100	200	100	0.00	96.04	96.04	3.96%
P3	200	400	200	97.25	298.69	201.44	0.72%
P2	400	4400	4000	298.80	4279.74	3980.94	0.48%
P4	4400	8400	4000	4279.85	8275.65	3995.80	0.10%
P5	8400	15400	7000	8275.76	15213.08	6937.32	0.90%

SJF_3.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P1	100	3100	3000	0.00	2891.54	2891.54	3.62%
P4	3100	3110	10	2900.74	2909.82	9.08	9.19%
P5	3110	3120	10	2909.94	2920.89	10.94	9.42%
P6	3120	7120	4000	2921.10	6862.80	3941.70	1.46%
P7	7120	11120	4000	6862.94	10903.97	4041.03	1.03%
P2	11120	16120	5000	10904.09	15913.46	5009.37	0.19%
P3	16120	23120	7000	15913.60	22978.51	7064.90	0.93%
P8	23120	32120	9000	22978.64	31646.77	8668.14	3.69%

SJF_4.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P1	0	3000	3000	0.00	3020.46	3020.46	0.68%
P2	3000	4000	1000	3020.62	4025.86	1005.24	0.52%
P3	4000	8000	4000	4026.01	8043.35	4017.35	0.43%
P5	8000	9000	1000	8043.53	9055.06	1011.52	1.15%
P4	9000	11000	2000	9069.23	11063.83	1994.61	0.27%

SJF_5.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P1	0	2000	2000	0.00	2017.27	2017.27	0.86%
P2	2000	2500	500	2017.42	2515.06	497.64	0.47%
P3	2500	3000	500	2515.18	3007.24	492.06	1.59%
P4	3000	3500	500	3023.27	3521.95	498.68	0.26%

TIME_MEASUREMENT.txt

process	start time	end time	expect exec time	my start time	my end time	my exec time	error rate
P0	0	500	500	0.00	512.15	512.15	2.43%
P1	1000	1500	500	999.37	1516.95	517.58	3.52%
P2	2000	2500	500	1967.75	2465.47	497.72	0.46%
P3	3000	3500	500	2971.35	3478.57	507.23	1.45%
P4	4000	4500	500	3965.82	4454.00	488.18	2.36%
P5	5000	5500	500	4951.90	5463.31	511.41	2.28%
P6	6000	6500	500	5929.39	6427.98	498.58	0.28%
P7	7000	7500	500	6945.59	7422.42	476.84	4.63%
P8	8000	8500	500	7884.82	8381.73	496.92	0.62%
P9	9000	9500	500	8890.84	9384.24	493.39	1.32%

Conclusion

- **Correctness**
  In my theoretical time assumption,there is no content switch and some hardware propagation delay, so each unit time is basicly the same.However,there might be some time error.But the error rate between theoretical execution time and real execution time aren still not smaller than 10%.The execution time is smaller than thereotical time because child process maybe run on the unavailable time before timer start. **My end time often be further from end time because previous process time error will affect the next one process,but the my exec time will not be affected by the previous one because if the process start earlier/later,it will end earlier/later.**

- **Work Loading**
  When CPU was busy or has other jobs,such like I/O input and output,the performance would be affected depending on the utilization of CPU.

- **Synchronization**
  Because we use two core to simulate parent and child process,maybe child will run on the unavailable time. So waitpid() is a key funtion in scheduler.c to make synchronization acheivable.