Operating System Project2 Report

Name: Wenhao Zhu
Student ID: 5130309717

1. Description of architecture

The main structure is the “simulator.cpp”, “memory.h” and “process.h”.

In the “process.h” file, I define the class of process, which has its id, name, start time, etc. It is just use for saving the process's information.

In the “memory.h” file, I define the class of memory, it is independent of the CPU scheduler. It provides some functions such as memory.append(), memory.find(), etc. All the replacement algorithm are embedded into the class. With different algorithm, memory is initialized in different modes, and in different mode, the functions are rather different. Thus the CPU does not need to consider how is the memory organized, it only needs to invoke the functions.

In the “simulator.cpp”. The first part read all kinds of file, get the parameters and initialize components. All the memory reference are read into a queue (every process have their own queue). The front of the queue refers to the address to be loaded next.

The main loop is a bit more complicated, and it deals with all the decisions in CPU. The loop will exit only when the ready queue, block queue, trace queue are all empty and there is no process is running. At the first of every cycle, the simulator will judge whether there is a process arrival, and push into the ready queue. The second, CPU will determine whether there is doing a context switch. If true, the scheduling part will do nothing, and enter the replacement part. Otherwise, it will determine whether there is a process running. If nothing is run and the ready queue is not empty, a new process will be loaded. If there is no running process, and the ready queue is empty, then this cycle will be idle cycle. If there is a running process, then we will determine whether there is a page fault. The third part Is about replacement, whether the paging is done, if done move the blocked process back to ready queue.

2. About decision policy

1) Do new processes go at the head of the ready list or the end?

The process goes at the end of the ready queue.

2) When a process starts up after a page fault, is it guaranteed to have the faulting page in memory, or could it fault again?

The new process could fault again, after dealing with the page fault (1000 cycles), the blocked process is just push at the last of the ready queue. If the page size is too small, or the designed replacement algorithm, when it reaches the head of ready queue, the page required is not sure to be in the memory. Thus there may be page fault again.

3) Do new processes or processes returning from a page fault get priority?

No, the scheduling algorithm is just simple Round-Robin algorithm, and doesn't match the true circumstances in reality. The process will not get priority, but go to the end of ready queue instead.

4) When there should be a context switch?

Except when a process is stopped by nature, there will be a context switch. Even the next process is itself.

5) If in one cycle, one process arrive and one process is awake from the block queue, which one will be the higher priority?

As I have mentioned before, at the first of every cycle, the simulator will judge whether there is a process arrival. And at the last part of the simulator it will deal with the replacement. So the arrival process will be put into queue earlier.

6) When will the context switch run? After the process is blocked or when a new process is loaded?

The context switch modifier will be changed after a new process is read, so when the process is blocked for 1000 cycle, we will not run in context switch at once.

3. Other tricky design for Simulation

In order to improve the performance of the simulator, to reduce the running time (in reality). I rewrite the simulator with c++ and using the STL library. And in the FIFO an second chance algorithm, I apply for a 256 space array for every space (because the memory trace has only two numbers which means the page from 0 to 255), then I can locate the page just by two keys: process name (id) and memory trace name, so either the find or the replace function need only O (1). It will be much faster than other data structure, but is needs more memory space (in reality).

4. Experiment1 Comparison of three replacement algorithms

Small scale data: 10 processes
Memory size: 500 pages
Quantum: 10000 cycles
(These test data can all been found in test data folder)

1) FIFO
Total time: 2041.19
Total cycle: 204118846
Total idle cycle: 202624245
Total page faults: 203601
2) LRU
Total time: 28.5325
Total cycle: 2853254
Total idle cycle: 1560385
Total page faults: 1869
3) SECOND CHANCE
Total time: 31.2949
Total cycle: 3129494
Total idle cycle: 1836333
Total page faults: 2161

Large scale data: 100 processes
Memory size: 500 pages
Quantum: 10000 cycles

(These test data can all been found in test data folder)

1) FIFO
Total time: 71182.2
Total cycle: 7118223999
Total idle cycle: 7098254432
Total page faults: 7117573
2) LRU
Total time: 61524.3
Total cycle: 6152427167
Total idle cycle: 6133423481
Total page faults: 6151692
3) SECOND CHANCE
Total time: 64292.2
Total cycle: 6429215364
Total idle cycle: 6409934880
Total page faults: 6428490

Analysis:
When the memory size is small, there are no obvious difference between these algorithms. Either algorithm is very slow and causes much page faults. When the size is too big, there is also no difference. Only when the memory size is proper, we can see the difference. According to the experiments, LRU algorithm's performance is better, and second-chance is a little better than FIFO. But it depends on the input data. When using big scale data, the LRU seems better. To sum up, LRU is the best among these three algorithms.

5. Experiment2 Comparison of quantum time with second-chance algorithm

Small scale data: 10 processes
Memory size: 500 pages
(These test data can all been found in test data folder)

1) Quantum =10000 cycles
Total time: 31.2949
Total cycle: 3129494
Total idle cycle: 1836333
Total page faults: 2161

2) Quantum = 50000 cycles
Total time: 40.5579
Total cycle: 4055793
Total idle cycle: 2761930
Total page faults: 2863

3) Quantum=200000 cycles
Total time: 43.2726

Total cycle: 4327259
Total idle cycle: 3033181
Total page faults: 3078

Analysis:

The bigger the quantum is, the more page faults will be. But it will not be so obvious. As quantum increase, a process will be executed longer. However, with a proper memory size, when the process frequently changed, many traces in the trace are loaded by the last or former processes, so there may be more page faults.

6. Experiment3 Comparison of memory size with second-chance algorithm

Large scale data: 100 processes
Quantum: 50000 cycles
(These test data can all been found in test data folder)

1) Memory size: 50 pages
Total time: 115675
Total cycle: 11567523958
Total idle cycle: 11543104587
Total page faults: 11567377

2) Memory size: 500 pages
Total time: 65859
Total cycle: 6585899264
Total idle cycle: 6566462259
Total page faults: 6585011

3) Memory size: 5000 pages
Total time: 193.467
Total cycle: 19346671
Total idle cycle: 6484704
Total page faults: 9973

Analysis:

The smaller the memory is, the more page faults will be. Because process switches so frequently, if the memory is small, the loaded trace will be replace soon, and the process comes again, there is still a page fault. So the program will be slow and will causes a lot of page faults. If we increase the page number, the page fault will decrease. After the memory size is equal to the sum of all memory traces, the page fault will reach the fewest. After that, even the memory size increases, the page fault will not decrease.

7. My own algorithm

1) Second-chance LRU algorithm (mode 3)	*2ch-lru*	*(abandoned)*
2) Lucky algorithm (mode 4)	*lucky*	*(abandoned)*
3) Communism algorithm (mode 5)	*cmu*	*(recommended)*

Description:

Communism algorithm is general FIFO, but I divide the whole memory into 10 parts, each process can only use one part , which is decide by process id mod 10. This algorithm can be good especially when the process are of similar number of memory trace, otherwise the overhead, or the waste of memory is a problem. It may be better than FIFO or second chance sometimes, for not all the process can preempt the memory space. One can only hold part of the resources, which may leads to a similar of better performance.

The algorithm still needs to be improved, for instance, the dynamic distribution of the memory space, or change FIFO to LRU. But that needs some time, maybe I’ll finish it in the summer holiday. I firmly believe that it can be improved a lot.