ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
Khoa học - Kỹ thuật Máy tính

HỆ ĐIỀU HÀNH

Group: NNHTV

ASSIGNMENT #2: Simple Operating System

GVHD: Hoàng Lê Hải Thanh

SV thực hiện: Huỳnh Quang Huy – 1812355
Trương Hoài Nam – 1813175
Nguyễn Hồng Nhân – 1813330
Trần Văn Tâm – 1713057
Hoàng Vương – 1811345

TP. HỒ CHÍ MINH, THÁNG 06/2020

Mục lục

1 Description of member's Work

Member	Description work
Huỳnh Quang Huy	Implement functions: get_proc() in sched.c Draw grantt chart for sched_0, sched_1
Trương Hoài Nam	Implement functions: enqueue() and dequeue() in sched.c Explanation memory management, Report
Nguyễn Hồng Nhân	Implement functions: enqueue() in sched.c Show the status of RAM after each memory allocation and deallocation function call m0
Trần Văn Tâm	Answer the questions in section Scheduler, Memory Management
Hoàng Vương	Implement functions: translate() and get_page_table() in mem.c Show the status of RAM after each memory allocation and deallocation function call in m1

2 Scheduler

2.1 Question

QUESTION: What is the advantage of using priority feedback queue in comparison with other scheduling algorithms you have learned?

Answer:

Trước tiên, ta cần hiểu về mục đích của các giải thuật scheduling để thấy được giải thuật priority feedback queue đã tác động ra sao đến các tiêu chí đánh giá. Tại một thời điểm trong memory sẽ có nhiều process, giả sử OS có 1 CPU (1 processor) thì tại mọi thời điểm đều thực thi 1 processor. Do đó, cần phải phân loại và lựa chọn process để thực thi sao cho hiệu quả nhất với các tiêu chí scheduling.

Các tiêu chí đó gồm: CPU utilization, Throughput, Turnaround-time, Waiting time, Average turn-around time, Respond time.

Giải thuật priority feedback queue là gì?

Priority feedback queue dựa trên giải thuật multilevel feedback queue được dùng trên Linux kernel.

Vậy giải thuật multilevel feedback queue (MFQ)là gì?

Trong hệ thống Multilevel Feedback Queue, scheduler có thể di chuyển process giữa các queue tùy theo đặc tính của nó được quan sát, thay đổi priority của process.

Ví dụ:

– Nếu một process sử dụng CPU quá lâu, nó sẽ bị di chuyển sang queue có priority thấp hơn.

– Nếu một process chờ qua lâu trong queue có priority thấp, nó sẽ được di chuyển lên queue có priority cao hơn (aging, giúp tránh starvation).

Giải thuật priority feedback queue đã áp dụng MFQ ra sao?

- Dùng hai queue có priority: ready queue và run queue, ready queue có priority cao hơn run queue nên được CPU thực thi trước. Khi CPU chuyển sang slot tiếp theo, nó sẽ tìm kiếm process trong ready queue:

+ Ready queue là 1 priority queue kế thừa 1 phần Priority Scheduling (PS), với mọi thời điểm CPU nhận process có priority cao nhất từ ready queue. Đó là điểm quan trọng trong giải thuật priority feedback queue.

+ Run queue: chứa các process đang chờ để tiếp tục thực thi sau khi hết slot của nó mà chưa hoàn tất quá trình của mình.

Khi nào thăng cấp một process?

Khi ready queue rỗi là empty thì Các process ở run queue đưa sang ready queue để xét slot tiếp theo.

Khi nào giáng cấp một process?

Khi process ở ready queue chạy hết time quantum được cho trước sẽ chuyển xuống run queue. Cả hai queue đều là priority queue, mức độ priority dựa trên mức độ priority của process trong queue.

Quy trình mà giải thuật priority feedback queue chạy:

- PCB của process được push vào ready queue và đợi CPU (enqueue).

- CPU chạy các process này theo round-robin style.

- Mỗi process được phép chạy tối đa trong khoảng time nhất định đã cho trước.

- Sau đó, CPU sẽ phải pause process đó và push đến run queue (1).

- CPU tiếp tục lấy process khác từ ready queue và tiếp tục chạy (2) (dequeue) CPU sẽ không đưa các process đã pause ở trên (1) trong run queue trở lại ready queue. Hậu quả xảy ra là ready queue sẽ bị empty do quá trình (2) liên tục xảy ra khiến ready queue 1 lúc nào đó sẽ không còn process.

Giải pháp:

Nếu ready queue bị empty thì dùng giải thuật priority feedback queue, scheduler này sẽ chuyển các process đang đợi trong run queue trở lại ready queue trước khi CPU nhận process chạy từ ready queue để quá trình (2) tiếp tục diễn ra, CPU chạy lại các process cũ đã bị pause (3):

+ (3) thể hiện rõ từ "feedback" trong giải thuật này.

+ (1)->(2)->(1)->(2)...->(3)... Điều này thể hiện rõ round robin "style".

Các giải thuật scheduling khác mà ta đã học có điểm bất lợi gì?

- **First Come First Served (FCFS)**:

- Do dùng chiến lược nonpreemptive trong process nên nếu 1 process bắt đầu thì CPU thực thi process cho đến khi nó kết thúc.

- Nếu process có CPU burst dài, thì các process phía sau(có thể là các process có CPU burst ngắn) trong queue phải đợi lâu -> Không công bằng cho các process.

- **Shortest Job First (SJF)**:

- Cần phải ước lượng thời gian cần CPU tiếp theo của process trước trong khi điều này là thường là không thể trong thực tế.

- Process có CPU burst dài có waiting time nhiều hơn hoặc trì hoãn vô hạn định khi đồng loạt có nhiều process có CPU burst ngắn vào queue->starvation.

- **Round Robin (RR)**:

- Throughput thường lớn phụ thuộc time quantum,nếu time quantum quá lớn: RR->FCFS.

- Nếu time quantum quá nhỏ, context switch của CPU sẽ tăng nhiều gây OS overhead, phí tổn-> Giảm CPU utilization.

- **Priority Scheduling (PS)**:

- Phải có scheduler với các process có priority ngang nhau.

- Các process có priority thấp hơn sẽ có thể không có cơ hội để thực thi -> starvation.

Giải thuật priority feedback queue(PFQ) khắc phục bất lợi trên như thế nào?

Các tiêu chí được thỏa mãn:

- Respond time ngắn hơn: Thuật toán này dựa trên Multilevel Feedback Queue, sử dụng hai queue khác nhau và chuyển các process qua lại giữa các queue đến khi process được hoàn tất.

- Turnaround time được optimize hơn: không như SJF là cần phải biết running time của process mà vẫn đảm bảo được turnaround time.PFQ chạy process theo time quantum sau đó thay đổi priority của process. Do đó, nó học hỏi từ hành vi quá khứ của process và sau đó dự đoán hành vi tương lai. Bằng cách này, PFQ cố gắng chạy process ngắn trước nên optimize được turnaround time.

- Nó cũng flexible hơn các giải thuật scheduling đã được học phía trên vì nó sử dụng ý tưởng của một số giải thuật khác gồm:

  + Priority Scheduling - mỗi process mang một priority để thực thi.

  + Multilevel Queue - sử dụng nhiều mức queue chocác process.

  + Round Robin - sử dụng quantum time cho các process thực thi.

- Đảm bảo tính công bằng cho các process đều được thực thi nhờ áp dụng Round Robin "style".

- Tránh starvation: Vì PFQ cũng cần ưu tiên các process có priority cao nên nếu không có các queue khác nhau thì process có priority thấp sẽ không đến lượt mình, nó vẫn được thực thi trước các process có priority cao hơn sau khi đã xong slot của mình (priority cao hơn chuyển sang run queue).

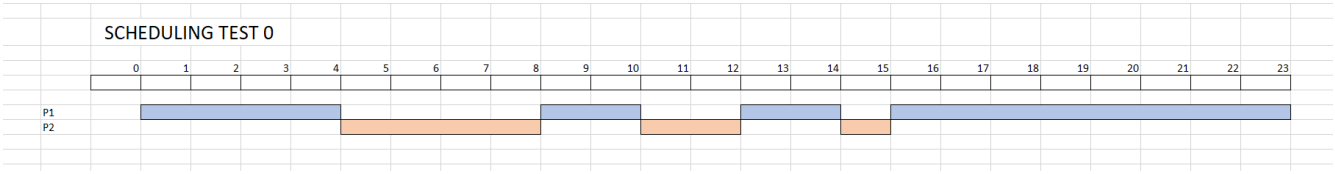
2.2 Gantt Diagrams

REQUIREMENT: Draw Gantt diagram describing how processes are executed by the CPU.

Dưới đây là giản đồ Gantt cho trường hợp trong .... trong source code.

TEST 0:

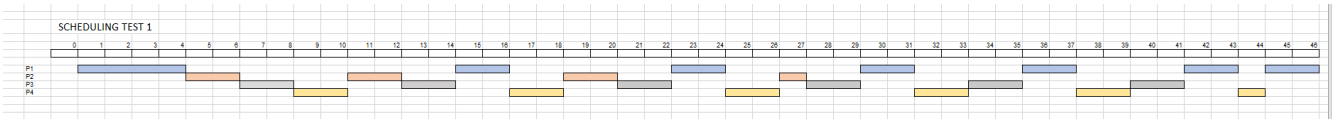
Trong test này, CPU xử lý trên 2 process p1 và p2 trong 23 time slot, như lượt đồ sau:



Hình 1: *Lược đồ Gantt CPU thực thi các processes - test 0*

TEST 1:

Trong test này, CPU xử lý trên 4 process p1, p2, p3 và p4 trong 46 time slot, như lượt đồ sau:



Hình 2: *Lược đồ Gantt CPU thực thi các processes - test 1*

2.3 Implementation

2.3.1 Priorty Queue

Hàng đợi ưu tiên có MAX_QUEUE_SIZE là 10 - xử lý không quá 10 process, do đó dùng vòng lặp để xử lý chức năng mà một hàng đợi ưu tiên cần có.
Cụ thể:

+ *enqueue()*: Đưa một process mới vào hàng đợi nếu sẵn sàng (hàng đợi còn trống).

+ *dequeue()*: Lấy process với ưu tiên cao nhất trong hàng đợi, cập nhật trạng thái queue khi lấy ra phần tử.

Dưới đây là phần hiện thực:

```
void enqueue(struct queue_t * q, struct pcb_t * proc) {
      /* TODO: put a new process to queue [q] */
      if(q -> size < MAX_QUEUE_SIZE){
            q -> proc[q->size] = proc;
            q -> size +=1;
      }
}

struct pcb_t * dequeue(struct queue_t * q) {
      /* TODO: return a pcb whose prioprity is the highest
      * in the queue [q] and remember to remove it from q
      * */
      struct pcb_t * proc_t;
```

```
    if(empty(q)){
        return NULL;
    }
    else{
        int max_pri = 0;

        for(int i = 1; i < q -> size; i++){
            if(q -> proc[i] -> priority > q -> proc[max_pri] -> priority){
                max_pri = i;
            }
        }
        proc_t = q -> proc[max_pri];
        q -> size -= 1;
        q -> proc[max_pri] = q -> proc[q->size];
        return proc_t;
    }
}
```

2.3.2 Scheduler

Scheduler có chức năng quản lý việc cập nhật các process sẽ được thực thi cho CPU, cụ thể là quản lý 2 hàng đợi ready và run như ở trên đã mô tả.

Cụ thể:

+ *get_proc()*: Trả về process của một quá trình ở hàng đợi sẵn sàng (ready). Nếu hàng đợi trống tại thời điểm hàm được gọi, cập nhật lại hàng đợi bằng các process đang chờ cho các slot tiếp theo trong hàng đợi run. Ngược lại, ta tìm ra process có độ ưu tiên cao từ hàng đợi này.

Dưới đây là phần hiện thực:

```
struct pcb_t * get_proc(void) {
    struct pcb_t * proc = NULL;
    /*TODO: get a process from [ready_queue]. If ready queue
    * is empty, push all processes in [run_queue] back to
    * [ready_queue] and return the highest priority one.
    * Remember to use lock to protect the queue.
    * */
    pthread_mutex_lock(&queue_lock);
    if(ready_queue.size==0)
    {
        for(int i=0; i<run_queue.size; i++)
        {
            enqueue(&ready_queue,run_queue.proc[i]);
            run_queue.proc[i]=NULL;
        }
        run_queue.size=0;
    }
    proc = dequeue(&ready_queue);
    pthread_mutex_unlock(&queue_lock);
    return proc;
}
```

3 Memory Management

3.1 Question

QUESTION: What is the advantage and disadvantage of segmentation with paging?

Answer:

Để biết được ưu nhược của segmentation với paing, ta cần hiểu chúng trước Segmentation với paging là cơ chế VME (Virtual Memory Engine) dùng để quản lý memory.

VME hoạt động như thế nào?

VME sẽ cô lập từng memory space của từng process riêng ra. Dù RAM (RAM trong assignment là RAM ảo) được shared để nhiều process chạy nhưng mỗi process nên không biết sự tồn tại của process khác. Để làm được điều này, mỗi process cần có 1 virtual memory space tương ứng và VME sẽ map và dịch nó sang physical address tương ứng.

Tuy nhiên không thể map toàn bộ virtual address space của 1 process sang address space của RAM (khi map sang physical address có thể trùng của process khác). Vì RAM được dùng chung bởi nhiều process và vùng memory cho 1 process không nên trùng với process khác.

Nên khi bắt đầu ta nên để segment table của process là empty tức là NULL (trong phần source code của assignment), và chỉ update khi process allocate hoặc deallocate memory. RAM được chia thành nhiều pages theo cơ chế Segmentation with Paging và OS allocate memory cho process theo từng page.

Tại sao dùng cơ chế Segmentation with Paging:

Kỹ thuật segmentation thõa mãn được nhu cầu thể hiện cấu trúc logic của program nhưng nó dẫn đến tình huống phải allocate các memory block có size khác nhau cho các segment trong physical memory. Điều này làm rắc rối vấn đề hơn rất nhiều so với việc allocate các page có size tĩnh và bằng nhau. Một giải pháp dung hoà là kết hợp cả hai kỹ thuật **Segmentation** và **Paging**: chúng ta tiến hành Paging các Segmentation.

Ý tưởng cơ chế Segmentation with Paging:

Address space là một tập các segment, mỗi segment được chia thành nhiều page. Khi một process được đưa vào OS, nó sẽ allocate cho process các page cần thiết để chứa đủ các segment của process. Nếu process yêu cầu 1 lượng memory nhỏ hơn page size, nó vẫn nhận được cả 1 page mới. Vì process dùng virtual address để truy cập RAM, ta nên để virtual address của các allocated pages là gần kề nhưng đối với physical address thì không cần gần kề.

Ví dụ:

1 process yêu cầu 2 page free trong RAM nhưng chúng ko gần kề: address của byte đầu tiên của 2 page lần lượt là 0x00000, 0x01000 (address của page là address của byte đầu tiên của page). Trong virtual address space, 2 page này phải gần kề mà address của byte đầu tiên là 0x00000, với offset = 5bits, nên page size = 32bytes, nên address của byte đầu tiên của page tiếp theo là 0x00020. Quá trình map:

```
Virtual      Ram
0x00000    0x00000
0x00020    0x01000
```

Từ đó, ra rút ra các lợi thế và bất lợi khi dùng cơ chế này là:

Lợi thế:

- Tiết kiệm tối ưu memory, sử dụng memory hiệu quả.

- Đơn giản việc allocate vùng nhớ không liên tục.

- Tận dụng ưu điểm của Paging và Segmentation và hạn chế khuyết điểm của chúng.

- Việc share dữ liệu giữa các tiến trình linh hoạt hơn qua chia sẻ 1 page hay 1 segment table.

- Khắc phục size của page quá lớn nhờ paging trong từng segment.

- Không xảy ra hiện thượng phân mảnh ngoại.

Bất lợi:

- Phân mảnh nội vẫn còn xảy ra của thuật toán Paging.

- Size của process bị giới hạn bởi size của physical memory.

- Khó có thể bảo trì nhiều process cùng lúc trong memory, và như vậy khó nâng cao mức độ multiprogramming của OS.

- Nếu segment table không được sử dụng thì không cần lưu page table.Page table cần nhiều memory space, nên sẽ không tốt cho system có RAM nhỏ.

⇒ Giảm memory space cần thiết khi lưu page table.

3.2 Show the status of RAM

Dưới đây là kết quả của quá trình ghi log sau mỗi lệnh allocation và deallocation trong chương trình, cụ thể ghi lại trạng thái của RAM trong chương trình ở mỗi bước.

test m0

```
------ MEMORY MANAGEMENT TEST 0 ----------------------------------
./mem input/proc/m0
=== PHYSICAL MEMORY AFTER ALLOCATION ====
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
========================================
=== PHYSICAL MEMORY AFTER ALLOCATION ====
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
```

```
001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
========================================
=== PHYSICAL MEMORY AFTER DEALLOCATION ===
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
========================================
=== PHYSICAL MEMORY AFTER ALLOCATION ====
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
========================================
=== PHYSICAL MEMORY AFTER ALLOCATION ====
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
========================================
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
003e8: 015
001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
03814: 066
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
NOTE: Read file output/m0 to verify your result
```

test m1

```
------ MEMORY MANAGEMENT TEST 1 ----------------------------------
./mem input/proc/m1
=== PHYSICAL MEMORY AFTER ALLOCATION ====
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
```

```
005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
========================================
=== PHYSICAL MEMORY AFTER ALLOCATION ====
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
========================================
=== PHYSICAL MEMORY AFTER DEALLOCATION ===
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
========================================
=== PHYSICAL MEMORY AFTER ALLOCATION ====
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
========================================
=== PHYSICAL MEMORY AFTER ALLOCATION ====
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
========================================
=== PHYSICAL MEMORY AFTER DEALLOCATION ===
002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
========================================
=== PHYSICAL MEMORY AFTER DEALLOCATION ===
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
```

```
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
=========================================
=== PHYSICAL MEMORY AFTER DEALLOCATION ===
=========================================
NOTE: Read file output/m1 to verify your result (your implementation should print nothing)
```

3.3 Explanation

3.3.1 Input of m0.txt

```
1 7
alloc 13535 0
alloc 1568 1
free 0
alloc 1386 2
alloc 4564 4
write 102 1 20
write 21 2 1000
```

Trên dòng đầu tiên, 1 là mức ưu tiên của process m0, và 7 là số lượng instruction của tệp đầu vào. 7 dòng tiếp theo là các câu lệnh allocate/dellocate hoặc sửa đổi RAM.

3.3.2 Output explanation

Bài tập lớn này được mô phỏng dựa trên hệ thống có địa chỉ kiến trúc là 20-bit, với 10 bits cuối là offset. Do đó, kích thước trang sẽ là $2^{10} = 1024$ bytes, và số lượng trang trong RAM là $2^{20}/2^{10} = 1024$ trang.

- Instruction 0: alloc 13535 0
  Quá trình m0 cấp phát (allocates) 13535 byte (14 trang từ 0 đến 13) và lưu địa chỉ ảo của trang đầu tiên cho thanh ghi 0 của quá trình. Mỗi dòng được in ra thể hiện thống kê của một trang được phân bổ trong RAM và có cấu trúc sau:
  page index: start_address - end_address - PID: pid (idx: index, nxt: next)
  - page index: index của page trong RAM (physical memory)
  - start_address: địa chỉ bắt đầu trong physical memory (hex format)
  - end_address: địa chỉ kết thúc trong physical memory (hex format)
  - pid: pid của process được cấp phát
  - index: index của page trong virtual memory
  - next: index của physical memory theo sau trong bộ nhớ ảo (virtual memory), là trang được phân bổ ngay sau trang này theo cùng instruction.
- Instruction 1: alloc 1568 1
  Quá trình m0 cấp phát (allocates) 1568 bytes (2 trang từ 14 đến 15) và lưu địa chỉ ảo của trang đầu tiên cho thanh ghi 1 của quá trình.
- Instruction 2: free 0
  Quá trình m0 giải phóng tất cả các trang được giữ bằng thanh ghi 0, là 14 trang từ 0 đến 13 trong RAM.
- Instruction 3: alloc 1386 2
  Quá trình m0 cấp phát 1386 bytes (2 trang từ 0 đến 1) và lưu địa chỉ ảo của trang đầu tiên cho thanh ghi 2 của quá trình.
- Instruction 4: alloc 4564 4
  Quá trình m0 cấp phát 4564 bytes (5 trang từ 2 đến 6) và lưu địa chỉ ảo của trang đầu tiên cho thanh ghi 4 của quá trình.

- Instruction 5: write 102 1 20
  Quá trình m0 ghi 102 (66 ở định dạng hex) vào bộ nhớ có địa chỉ vật lý được tính như sau:
  $physical\_address = base\_address + offset$
  - base_address: địa chỉ vật lý của byte đầu tiên được giữ bởi thanh ghi thứ nhất, có địa chỉ 0x03800
  - offset: 20 (14 ở dạng hex)

  Ở trường hợp này, địa chỉ vật lý là 0x03814.
- Instruction 6: write 21 2 1000
  Quá trình m0 ghi 21 (15 ở định dạng hex) vào bộ nhớ có địa chỉ vật lý 00000 (0x00000) + 1000 (0x3e8) = 0x3e8.

4 Overall

4.1 Synchronization

Trong file *mem.c*, hai hàm *read_mem()* và *write_mem()* truy cập vào *[_ram]* có thể xảy ra bất đồng bộ nên ta cần thêm một *mutex_lock*: *ram_lock* để giải quyết, khi đó hai hàm trên được chỉnh lại như sau:

- **Hàm read_mem()**

```
int read_mem(addr_t address, struct pcb_t * proc, BYTE * data) {
   addr_t physical_addr;
   if (translate(address, &physical_addr, proc)) {
      pthread_mutex_lock(&ram_lock);
      *data = _ram[physical_addr];
      pthread_mutex_unlock(&ram_lock);
      return 0;
   }else{
      return 1;
   }
}
```

- **Hàm write_mem()**

```
int write_mem(addr_t address, struct pcb_t * proc, BYTE data) {
   addr_t physical_addr;
   if (translate(address, &physical_addr, proc)) {
      pthread_mutex_lock(&ram_lock);
      _ram[physical_addr] = data;
      pthread_mutex_unlock(&ram_lock);
      return 0;
   }else{
      return 1;
   }
}
```

4.2 Result of simulation

```
------ MEMORY MANAGEMENT TEST 0 ----------------------------------
./mem input/proc/m0
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
003e8: 015
001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
```

```
004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
03814: 066
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
NOTE: Read file output/m0 to verify your result
------ MEMORY MANAGEMENT TEST 1 ----------------------------------
./mem input/proc/m1
NOTE: Read file output/m1 to verify your result (your implementation should print nothing)
------ SCHEDULING TEST 0 ------------------------------------------
./os sched_0
Time slot  0
Loaded a process at input/proc/s0, PID: 1
Time slot  1
CPU 0: Dispatched process 1
Time slot  2
Time slot  3
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 1
Time slot  4
Loaded a process at input/proc/s1, PID: 2
Time slot  5
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 2
Time slot  6
Time slot  7
CPU 0: Put process 2 to run queue
CPU 0: Dispatched process 2
Time slot  8
Time slot  9
CPU 0: Put process 2 to run queue
CPU 0: Dispatched process 1
Time slot 10
Time slot 11
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 2
Time slot 12
Time slot 13
CPU 0: Put process 2 to run queue
CPU 0: Dispatched process 1
Time slot 14
Time slot 15
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 2
Time slot 16
CPU 0: Processed 2 has finished
CPU 0: Dispatched process 1
Time slot 17
Time slot 18
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 1
Time slot 19
Time slot 20
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 1
Time slot 21
Time slot 22
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 1
```

```
Time slot 23
CPU 0: Processed 1 has finished
CPU 0 stopped

MEMORY CONTENT:
NOTE: Read file output/sched_0 to verify your result
------ SCHEDULING TEST 1 --------------------------------------------
./os sched_1
Time slot  0
Loaded a process at input/proc/s0, PID: 1
Time slot  1
CPU 0: Dispatched process 1
Time slot  2
Time slot  3
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 1
Time slot  4
Loaded a process at input/proc/s1, PID: 2
Time slot  5
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 2
Time slot  6
Loaded a process at input/proc/s2, PID: 3
Time slot  7
CPU 0: Put process 2 to run queue
CPU 0: Dispatched process 3
Loaded a process at input/proc/s3, PID: 4
Time slot  8
Time slot  9
CPU 0: Put process 3 to run queue
CPU 0: Dispatched process 4
Time slot 10
Time slot 11
CPU 0: Put process 4 to run queue
CPU 0: Dispatched process 2
Time slot 12
Time slot 13
CPU 0: Put process 2 to run queue
CPU 0: Dispatched process 3
Time slot 14
Time slot 15
CPU 0: Put process 3 to run queue
CPU 0: Dispatched process 1
Time slot 16
Time slot 17
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 4
Time slot 18
Time slot 19
CPU 0: Put process 4 to run queue
CPU 0: Dispatched process 2
Time slot 20
Time slot 21
CPU 0: Put process 2 to run queue
CPU 0: Dispatched process 3
Time slot 22
Time slot 23
CPU 0: Put process 3 to run queue
CPU 0: Dispatched process 1
Time slot 24
```

```
Time slot 25
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 4
Time slot 26
Time slot 27
CPU 0: Put process 4 to run queue
CPU 0: Dispatched process 2
Time slot 28
CPU 0: Processed 2 has finished
CPU 0: Dispatched process 3
Time slot 29
Time slot 30
CPU 0: Put process 3 to run queue
CPU 0: Dispatched process 1
Time slot 31
Time slot 32
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 4
Time slot 33
Time slot 34
CPU 0: Put process 4 to run queue
CPU 0: Dispatched process 3
Time slot 35
Time slot 36
CPU 0: Put process 3 to run queue
CPU 0: Dispatched process 1
Time slot 37
Time slot 38
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 4
Time slot 39
Time slot 40
CPU 0: Put process 4 to run queue
CPU 0: Dispatched process 3
Time slot 41
Time slot 42
CPU 0: Processed 3 has finished
CPU 0: Dispatched process 1
Time slot 43
Time slot 44
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 4
Time slot 45
CPU 0: Processed 4 has finished
CPU 0: Dispatched process 1
Time slot 46
CPU 0: Processed 1 has finished
CPU 0 stopped

MEMORY CONTENT:
NOTE: Read file output/sched_1 to verify your result
----- OS TEST 0 -------------------------------------------------
./os os_0
Time slot  0
Loaded a process at input/proc/p0, PID: 1
Time slot  1
CPU 1: Dispatched process 1
Time slot  2
Loaded a process at input/proc/p1, PID: 2
CPU 0: Dispatched process 2
```

```
Time slot  3
Loaded a process at input/proc/p1, PID: 3
Time slot  4
Loaded a process at input/proc/p1, PID: 4
Time slot  5
Time slot  6
Time slot  7
CPU 1: Put process 1 to run queue
CPU 1: Dispatched process 3
Time slot  8
CPU 0: Put process 2 to run queue
CPU 0: Dispatched process 4
Time slot  9
Time slot 10
Time slot 11
Time slot 12
Time slot 13
CPU 1: Put process 3 to run queue
CPU 1: Dispatched process 1
Time slot 14
CPU 0: Put process 4 to run queue
CPU 0: Dispatched process 3
Time slot 15
Time slot 16
Time slot 17
CPU 1: Processed 1 has finished
CPU 1: Dispatched process 2
Time slot 18
CPU 0: Processed 3 has finished
CPU 0: Dispatched process 4
Time slot 19
Time slot 20
Time slot 21
CPU 1: Processed 2 has finished
CPU 1 stopped
Time slot 22
CPU 0: Processed 4 has finished
CPU 0 stopped

MEMORY CONTENT:
000: 00000-003ff - PID: 01 (idx 000, nxt: -01)
001: 00400-007ff - PID: 02 (idx 000, nxt: 012)
002: 00800-00bff - PID: 02 (idx 000, nxt: 003)
009e7: 00a
003: 00c00-00fff - PID: 02 (idx 001, nxt: 004)
004: 01000-013ff - PID: 02 (idx 002, nxt: 005)
005: 01400-017ff - PID: 02 (idx 003, nxt: 006)
006: 01800-01bff - PID: 02 (idx 004, nxt: -01)
007: 01c00-01fff - PID: 01 (idx 000, nxt: 008)
01c14: 064
008: 02000-023ff - PID: 01 (idx 001, nxt: 009)
009: 02400-027ff - PID: 01 (idx 002, nxt: 010)
010: 02800-02bff - PID: 01 (idx 003, nxt: 011)
011: 02c00-02fff - PID: 01 (idx 004, nxt: -01)
015: 03c00-03fff - PID: 03 (idx 000, nxt: 016)
03de7: 00a
016: 04000-043ff - PID: 03 (idx 001, nxt: 017)
017: 04400-047ff - PID: 03 (idx 002, nxt: 018)
018: 04800-04bff - PID: 03 (idx 003, nxt: 019)
019: 04c00-04fff - PID: 03 (idx 004, nxt: -01)
```

```
020: 05000-053ff - PID: 04 (idx 000, nxt: 021)
051e7: 00a
021: 05400-057ff - PID: 04 (idx 001, nxt: 022)
022: 05800-05bff - PID: 04 (idx 002, nxt: 023)
023: 05c00-05fff - PID: 04 (idx 003, nxt: 024)
024: 06000-063ff - PID: 04 (idx 004, nxt: -01)
029: 07400-077ff - PID: 04 (idx 000, nxt: 030)
030: 07800-07bff - PID: 04 (idx 001, nxt: 031)
031: 07c00-07fff - PID: 04 (idx 002, nxt: 032)
032: 08000-083ff - PID: 04 (idx 003, nxt: -01)
033: 08400-087ff - PID: 03 (idx 000, nxt: 034)
034: 08800-08bff - PID: 03 (idx 001, nxt: 035)
035: 08c00-08fff - PID: 03 (idx 002, nxt: 036)
036: 09000-093ff - PID: 03 (idx 003, nxt: -01)
037: 09400-097ff - PID: 04 (idx 000, nxt: 038)
038: 09800-09bff - PID: 04 (idx 001, nxt: 039)
039: 09c00-09fff - PID: 04 (idx 002, nxt: 040)
040: 0a000-0a3ff - PID: 04 (idx 003, nxt: -01)
NOTE: Read file output/os_0 to verify your result
----- OS TEST 1 -------------------------------------------------
./os os_1
Time slot  0
Time slot  1
Loaded a process at input/proc/p0, PID: 1
CPU 0: Dispatched process 1
Time slot  2
Loaded a process at input/proc/s3, PID: 2
CPU 2: Dispatched process 2
Time slot  3
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 1
Time slot  4
Loaded a process at input/proc/m1, PID: 3
CPU 2: Put process 2 to run queue
CPU 2: Dispatched process 3
Time slot  5
CPU 3: Dispatched process 2
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 1
Time slot  6
Loaded a process at input/proc/s2, PID: 4
CPU 2: Put process 3 to run queue
CPU 2: Dispatched process 4
Time slot  7
CPU 3: Put process 2 to run queue
CPU 3: Dispatched process 2
CPU 1: Dispatched process 3
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 1
Loaded a process at input/proc/m0, PID: 5
Time slot  8
CPU 2: Put process 4 to run queue
CPU 2: Dispatched process 5
Time slot  9
CPU 3: Put process 2 to run queue
CPU 3: Dispatched process 4
CPU 1: Put process 3 to run queue
CPU 1: Dispatched process 2
CPU 0: Put process 1 to run queue
CPU 0: Dispatched process 3
```

```
Loaded a process at input/proc/p1, PID: 6
Time slot 10
CPU 2: Put process 5 to run queue
CPU 2: Dispatched process 1
Time slot 11
CPU 3: Put process 4 to run queue
CPU 3: Dispatched process 6
CPU 1: Put process 2 to run queue
CPU 1: Dispatched process 4
Loaded a process at input/proc/s0, PID: 7
CPU 0: Put process 3 to run queue
CPU 0: Dispatched process 7
Time slot 12
CPU 2: Processed 1 has finished
CPU 2: Dispatched process 2
Time slot 13
CPU 3: Put process 6 to run queue
CPU 3: Dispatched process 5
CPU 1: Put process 4 to run queue
CPU 1: Dispatched process 4
CPU 0: Put process 7 to run queue
CPU 0: Dispatched process 3
Time slot 14
CPU 2: Put process 2 to run queue
CPU 2: Dispatched process 6
Time slot 15
CPU 3: Put process 5 to run queue
CPU 3: Dispatched process 7
CPU 1: Put process 4 to run queue
CPU 1: Dispatched process 2
CPU 0: Processed 3 has finished
CPU 0: Dispatched process 5
Time slot 16
CPU 1: Processed 2 has finished
Loaded a process at input/proc/s1, PID: 8
CPU 2: Put process 6 to run queue
CPU 2: Dispatched process 4
CPU 1: Dispatched process 8
Time slot 17
CPU 3: Put process 7 to run queue
CPU 3: Dispatched process 6
CPU 0: Put process 5 to run queue
CPU 0: Dispatched process 7
Time slot 18
CPU 1: Put process 8 to run queue
CPU 1: Dispatched process 5
CPU 2: Put process 4 to run queue
CPU 2: Dispatched process 8
Time slot 19
CPU 3: Put process 6 to run queue
CPU 3: Dispatched process 4
CPU 0: Put process 7 to run queue
CPU 0: Dispatched process 7
CPU 1: Processed 5 has finished
CPU 1: Dispatched process 6
Time slot 20
CPU 2: Put process 8 to run queue
CPU 2: Dispatched process 8
Time slot 21
CPU 3: Processed 4 has finished
```

```
CPU 3 stopped
CPU 1: Put process 6 to run queue
CPU 1: Dispatched process 6
CPU 0: Put process 7 to run queue
CPU 0: Dispatched process 7
Time slot 22
CPU 2: Put process 8 to run queue
CPU 2: Dispatched process 8
Time slot 23
CPU 1: Processed 6 has finished
CPU 1 stopped
CPU 0: Put process 7 to run queue
CPU 0: Dispatched process 7
CPU 2: Processed 8 has finished
CPU 2 stopped
Time slot 24
Time slot 25
CPU 0: Put process 7 to run queue
CPU 0: Dispatched process 7
Time slot 26
Time slot 27
CPU 0: Put process 7 to run queue
CPU 0: Dispatched process 7
Time slot 28
CPU 0: Processed 7 has finished
CPU 0 stopped

MEMORY CONTENT:
000: 00000-003ff - PID: 06 (idx 000, nxt: 001)
001e7: 00a
001: 00400-007ff - PID: 06 (idx 001, nxt: 031)
002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
00814: 064
003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
007: 01c00-01fff - PID: 05 (idx 000, nxt: 008)
01fe8: 015
008: 02000-023ff - PID: 05 (idx 001, nxt: -01)
014: 03800-03bff - PID: 06 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 06 (idx 001, nxt: 016)
016: 04000-043ff - PID: 06 (idx 002, nxt: 017)
017: 04400-047ff - PID: 06 (idx 003, nxt: -01)
018: 04800-04bff - PID: 06 (idx 000, nxt: 020)
019: 04c00-04fff - PID: 01 (idx 000, nxt: -01)
020: 05000-053ff - PID: 06 (idx 001, nxt: 021)
021: 05400-057ff - PID: 06 (idx 002, nxt: 022)
022: 05800-05bff - PID: 06 (idx 003, nxt: -01)
024: 06000-063ff - PID: 05 (idx 000, nxt: 025)
06014: 066
025: 06400-067ff - PID: 05 (idx 001, nxt: -01)
031: 07c00-07fff - PID: 06 (idx 002, nxt: 032)
032: 08000-083ff - PID: 06 (idx 003, nxt: 033)
033: 08400-087ff - PID: 06 (idx 004, nxt: -01)
NOTE: Read file output/os_1 to verify your result
```

Tài liệu