CS 344
Operating Systems Lab

Assignment - 2B

Report

Team Members:

1. Braj Nandan Kalya - 190101029
2. Anurag Singhal - 190123010
3. Pallav Pandey - 190123044
4. Alokeveer Mondal - 190123003

Task 1

- The scheduler() function in proc.c selects the process for running on the basis of a defined scheduling algorithm.

```
void
scheduler(void)
{
  struct proc *p;
  struct cpu *c = mycpu();
  c->proc = 0;
```

- wakeup1() function fires when a process return from I/O.

```
653  static void
654  wakeup1(void *chan)
655  {
656    struct proc *p;
657
658    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
659      if(p->state == SLEEPING && p->chan == chan) {
660        p->state = RUNNABLE;
661        #ifdef DML
662        p->priority = 3; // relevant for DML - process waited for I\O, and now it's ready to run again
663        #endif
664      }
665  }
666
```

- Allocproc() function in proc.c fires when a new process is created. It allocates resources to the process.

```
//PAGEBREAK: 32
// Look in the process table for an UNUSED proc.
// If found, change state to EMBRYO and initialize
// state required to run in the kernel.
// Otherwise return 0.
static struct proc*
allocproc(void)
{
  struct proc *p;
  char *sp;

  acquire(&ptable.lock);

  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
    if(p->state == UNUSED)
      goto found;
```

- Scheduling takes place whenever
    1. In default, time quantum is completed for the current process
    2. In FCFS, the current process is completed.
    3. In SML, either process is completed or a higher priority process is ready.
    4. In DML, either process is completed or a higher priority process is ready or if time quantum is over.

Policy 1: Default Policy

```
451  scheduler(void)
452  {
453    struct proc *p;
454    struct cpu *c = mycpu();
455    c->proc = 0;
456    int index1 = 0;
457    int index2 = 0;
458    int index3 = 0;
459    int x = index1;
460    index1 = x;
461    x = index2;
462    index2 = x;
463    x = index3;
464    index3 = x;
465
466
467    for(;;){
468      // Enable interrupts on this processor.
469      sti();
470
471      // Loop over process table looking for process to run.
472      acquire(&ptable.lock);
473      #ifdef DEFAULT
474      for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
475        if(p->state != RUNNABLE)
476          continue;
477
478        // Switch to chosen process.  It is the process's job
479        // to release ptable.lock and then reacquire it
480        // before jumping back to us.
481        c->proc = p;
482        switchuvm(p);
483        p->state = RUNNING;
484
485        swtch(&(c->scheduler), p->context);
486        switchkvm();
487
488        // Process is done running for now.
489        // It should have changed its p->state before coming back.
490        c->proc = 0;
491      }
492      #else
```

Represents the scheduling policy currently implemented at xv6 (with the only difference being the newly defined QUANTA).

Policy 2: First come - First Served

```
493      #ifdef FCFS
494        struct proc *minP = NULL;
495          for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
496            if(p->state == RUNNABLE){
497              if (minP!=NULL){
498                if(p->ctime < minP->ctime)
499                  minP = p;
500              }
501              else
502                minP = p;
503            }
504          }
505          if (minP!=NULL){
506            p = minP;//the process with the smallest creation time
507            c->proc = p;
508            switchuvm(p);
509            p->state = RUNNING;
510            swtch(&c->scheduler, myproc()->context);
511            switchkvm();
512            // Process is done running for now.
513            // It should have changed its p->state before coming back.
514            c->proc = 0;
515          }
516      #else
```

Represents a non-preemptive policy that selects the process with the lowest creation time. The process runs until it no longer needs CPU time (IO / yield / block).

Policy 3: Multi-level queue scheduling

```
361
362  #ifdef SML
363  struct proc* findreadyprocess(int *index1, int *index2, int *index3, uint *priority) {
364    int i;
365    struct proc* proc2;
366  notfound:
367    for (i = 0; i < NPROC; i++) {
368      switch(*priority) {
369        case 1:
370          proc2 = &ptable.proc[(*index1 + i) % NPROC];
371          if (proc2->state == RUNNABLE && proc2->priority == *priority) {
372            *index1 = (*index1 + 1 + i) % NPROC;
373            return proc2; // found a runnable process with appropriate priority
374          }
375        case 2:
376          proc2 = &ptable.proc[(*index2 + i) % NPROC];
377          if (proc2->state == RUNNABLE && proc2->priority == *priority) {
378            *index2 = (*index2 + 1 + i) % NPROC;
379            return proc2; // found a runnable process with appropriate priority
380          }
381        case 3:
382          proc2 = &ptable.proc[(*index3 + i) % NPROC];
383          if (proc2->state == RUNNABLE && proc2->priority == *priority){
384            *index3 = (*index3 + 1 + i) % NPROC;
385            return proc2; // found a runnable process with appropriate priority
386          }
387      }
388    }
389    if (*priority == 1) {//did not find any process on any of the prorities
390      *priority = 3;
391      return 0;
392    }
393    else {
394      *priority -= 1; //will try to find a process at a lower priority
395      goto notfound;
396    }
397    return 0;
398  }
399  #endif
```

Represents a preemptive policy that includes a three priority queues.

This function chooses the process to be run according to their priority.

```
517       #ifdef SML
518       uint priority = 3;
519       p = findreadyprocess(&index1, &index2, &index3, &priority);
520       if (p == 0) {
521         release(&ptable.lock);
522         continue;
523       }
524       c->proc = p;
525       switchuvm(p);
526       p->state = RUNNING;
527       swtch(&c->scheduler, myproc()->context);
528       switchkvm();
529       c->proc = 0;
530       #else
```

In this scheduling policy the scheduler will select a process from a lower queue only if no process is ready to run at a higher queue. Moving between priority queues is only available via a system call. Priority 3 is the highest priority.

Policy 4: Dynamic Multi-level queue scheduling

```
401  #ifdef DML
402  /*
403    this method will find the next process to run
404  */
405  struct proc* findreadyprocess(int *index1, int *index2, int *index3, uint *priority) {
406    int i;
407    struct proc* proc2;
408  notfound:
409    for (i = 0; i < NPROC; i++) {
410      switch(*priority) {
411        case 1:
412          proc2 = &ptable.proc[(*index1 + i) % NPROC];
413          if (proc2->state == RUNNABLE && proc2->priority == *priority) {
414            *index1 = (*index1 + 1 + i) % NPROC;
415            return proc2; // found a runnable process with appropriate priority
416          }
417        case 2:
418          proc2 = &ptable.proc[(*index2 + i) % NPROC];
419          if (proc2->state == RUNNABLE && proc2->priority == *priority) {
420            *index2 = (*index2 + 1 + i) % NPROC;
421            return proc2; // found a runnable process with appropriate priority
422          }
423        case 3:
424          proc2 = &ptable.proc[(*index3 + i) % NPROC];
425          if (proc2->state == RUNNABLE && proc2->priority == *priority){
426            *index3 = (*index3 + 1 + i) % NPROC;
427            return proc2; // found a runnable process with appropriate priority
428          }
429      }
430    }
431    if (*priority == 1) {//did not find any process on any of the prorities
432      *priority = 3;
433      return 0;
434    }
435    else {
436      *priority -= 1; //will try to find a process at a lower priority
437      goto notfound;
438    }
439    return 0;
440  }
441  #endif
```

Represents a preemptive policy similar to Policy 3. The difference is that the process cannot manually change its priority.

Running the full quanta will result in a decrease of priority by 1.

This function chooses the process to be run according to their priority.

```
531        #ifdef DML
532        uint priority = 3;
533        p = findreadyprocess(&index1, &index2, &index3, &priority);
534        if (p == 0) {
535          release(&ptable.lock);
536          continue;
537        }
538        c->proc = p;
539        switchuvm(p);
540        p->state = RUNNING;
541        p->tickcounter = 0;
542        swtch(&c->scheduler, myproc()->context);
543        switchkvm();
544        c->proc = 0;
545        #endif
546        #endif
547        #endif
548        #endif
549        release(&ptable.lock);
550
551      }
552    }
```

```
767  void resetticksycle(int *counter) {
768    acquire(&ptable.lock);
769    *counter = 0;
770    release(&ptable.lock);
771  }
772
```

Function created to reset tick counter for each process. It is called when time quantum of a proess is completed.

```
737  int set_prio(int priority) {
738    if (priority < 1 || priority > 3)
739      return 1;
740    acquire(&ptable.lock);
741    myproc()->priority = priority;
742    release(&ptable.lock);
743    return 0;
744  }
745
```

This function sets priority of the processes.

```
773  void decpriority(void) {
774    // acquire(&ptable.lock);
775    myproc()->priority = myproc()->priority == 1 ? 1 : myproc()->priority - 1;
776    // release(&ptable.lock);
777  }
778
779  int inctickcounter() {
780    int res;
781    acquire(&ptable.lock);
782    res = ++myproc()->tickcounter;
783    release(&ptable.lock);
784    return res;
785  }
```

- decpriority() decreases the priority of the process by 1 if not already = 1. It is called when the time quantum of a process is over in DML policy.

Following were the changes in makefile.

```
73
74  ifndef SCHEDFLAG
75  SCHEDFLAG := DEFAULT
76  endif
77
78  CC = $(TOOLPREFIX)gcc
79  AS = $(TOOLPREFIX)gas
80  LD = $(TOOLPREFIX)ld
81  OBJCOPY = $(TOOLPREFIX)objcopy
82  OBJDUMP = $(TOOLPREFIX)objdump
83  CFLAGS = -fno-pic -static -fno-builti
84  CFLAGS += -D $(SCHEDFLAG)
```

```
187         _wc\
188         _zombie\
189         _sanity\
190         _SMLsanity\
191
```

Task 2

yield() and set_prio() system calls were added.

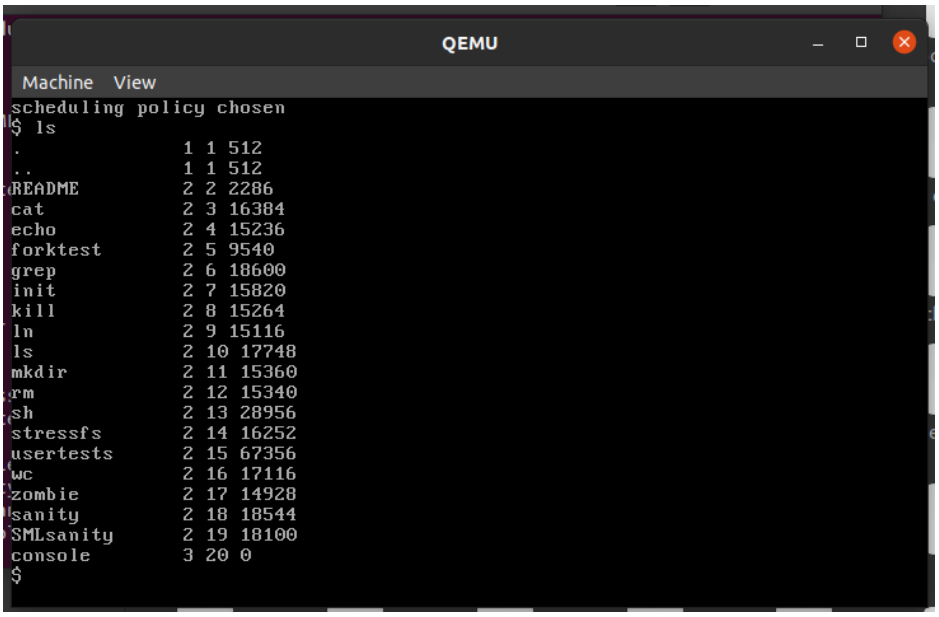
```
23  #define SYS_history  22
24  #define SYS_wait2 23
25  #define SYS_set_prio 24
26  #define SYS_yield  25
```

```
11  #define LOGSIZE      (MAXOPB
12  #define NBUF         (MAXOPB
13  #define FSSIZE       1000  /
14  #define QUANTA       5
15
```

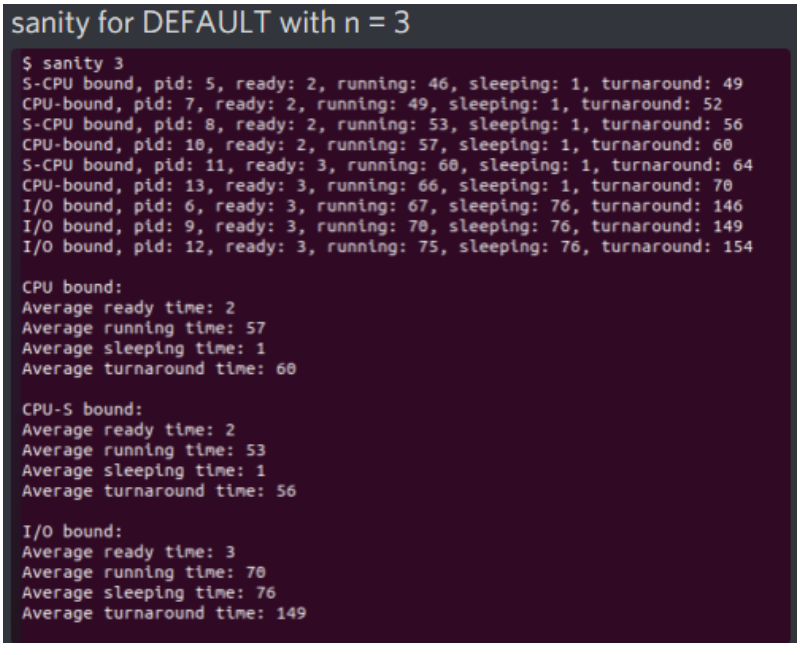
```
122  int sys_set_prio(void) {
123    int priority;
124    argint(0, &priority);
125    return set_prio(priority);
126  }
127
128  int sys_yield(void) {
129    yield();
130    return 0;
131  }
```

Task 3

1. Sanity for n = 3,
    - sanity for DEFAULT
    - sanity for FCFS
    - sanity for SML
    - sanity for DML
2. SMLsanity with n = 10



Sanity and SMLsanity, both are visible in ‘ls’.



sanity for FCFS with n = 3

```
$ sanity 3
CPU-bound, pid: 4, ready: 0, running: 7, sleeping: 0, turnaround: 7
S-CPU bound, pid: 5, ready: 0, running: 11, sleeping: 0, turnaround: 11
CPU-bound, pid: 7, ready: 0, running: 14, sleeping: 0, turnaround: 14
S-CPU bound, pid: 8, ready: 0, running: 16, sleeping: 0, turnaround: 16
CPU-bound, pid: 10, ready: 0, running: 21, sleeping: 0, turnaround: 21
S-CPU bound, pid: 11, ready: 0, running: 23, sleeping: 0, turnaround: 23
I/O bound, pid: 6, ready: 0, running: 26, sleeping: 79, turnaround: 105
I/O bound, pid: 9, ready: 0, running: 28, sleeping: 79, turnaround: 107
I/O bound, pid: 12, ready: 0, running: 33, sleeping: 79, turnaround: 112

CPU bound:
Average ready time: 0
Average running time: 14
Average sleeping time: 0
Average turnaround time: 14

CPU-S bound:
Average ready time: 0
Average running time: 16
Average sleeping time: 0
Average turnaround time: 16

I/O bound:
Average ready time: 0
Average running time: 29
Average sleeping time: 79
Average turnaround time: 108

$
```

sanity for DML with n = 3

```
$ sanity 3
CPU-bound, pid: 4, ready: 0, running: 11, sleeping: 1, turnaround: 12
S-CPU bound, pid: 5, ready: 0, running: 16, sleeping: 1, turnaround: 17
CPU-bound, pid: 7, ready: 0, running: 20, sleeping: 1, turnaround: 21
S-CPU bound, pid: 8, ready: 0, running: 27, sleeping: 1, turnaround: 28
CPU-bound, pid: 10, ready: 0, running: 32, sleeping: 1, turnaround: 33
S-CPU bound, pid: 11, ready: 0, running: 36, sleeping: 1, turnaround: 37
I/O bound, pid: 6, ready: 0, running: 38, sleeping: 68, turnaround: 106
I/O bound, pid: 9, ready: 0, running: 40, sleeping: 69, turnaround: 109
I/O bound, pid: 12, ready: 0, running: 43, sleeping: 69, turnaround: 112

CPU bound:
Average ready time: 0
Average running time: 21
Average sleeping time: 1
Average turnaround time: 22

CPU-S bound:
Average ready time: 0
Average running time: 26
Average sleeping time: 1
Average turnaround time: 27

I/O bound:
Average ready time: 0
Average running time: 40
Average sleeping time: 68
Average turnaround time: 108

$
```

sanity for SML with n = 3

```
$ sanity 3
S-CPU bound, pid: 5, ready: 2, running: 10, sleeping: 1, turnaround: 13
CPU-bound, pid: 7, ready: 2, running: 13, sleeping: 1, turnaround: 16
S-CPU bound, pid: 8, ready: 2, running: 16, sleeping: 1, turnaround: 19
CPU-bound, pid: 10, ready: 2, running: 17, sleeping: 1, turnaround: 20
S-CPU bound, pid: 11, ready: 2, running: 21, sleeping: 1, turnaround: 24
CPU-bound, pid: 13, ready: 2, running: 24, sleeping: 1, turnaround: 27
I/O bound, pid: 6, ready: 2, running: 29, sleeping: 80, turnaround: 111
I/O bound, pid: 9, ready: 2, running: 34, sleeping: 80, turnaround: 116
I/O bound, pid: 12, ready: 2, running: 38, sleeping: 80, turnaround: 120

CPU bound:
Average ready time: 2
Average running time: 18
Average sleeping time: 1
Average turnaround time: 21

CPU-S bound:
Average ready time: 2
Average running time: 15
Average sleeping time: 1
Average turnaround time: 18

I/O bound:
Average ready time: 2
Average running time: 33
Average sleeping time: 80
Average turnaround time: 115

$
```

SMLsanity with n = 10

```
$ SMLsanity 10
Priority 3, pid: 15, ready: 2, running: 86, sleeping: 82, turnaround: 170
Priority 1, pid: 16, ready: 2, running: 89, sleeping: 82, turnaround: 173
Priority 2, pid: 17, ready: 2, running: 94, sleeping: 82, turnaround: 178
Priority 3, pid: 18, ready: 2, running: 98, sleeping: 82, turnaround: 182
Priority 1, pid: 19, ready: 2, running: 99, sleeping: 82, turnaround: 183
Priority 2, pid: 20, ready: 2, running: 103, sleeping: 82, turnaround: 187
Priority 3, pid: 21, ready: 2, running: 105, sleeping: 82, turnaround: 189
Priority 1, pid: 22, ready: 2, running: 109, sleeping: 82, turnaround: 193
Priority 2, pid: 23, ready: 2, running: 113, sleeping: 82, turnaround: 197
Priority 3, pid: 24, ready: 2, running: 116, sleeping: 82, turnaround: 200
Priority 1, pid: 25, ready: 2, running: 117, sleeping: 82, turnaround: 201
Priority 2, pid: 26, ready: 2, running: 120, sleeping: 82, turnaround: 204
Priority 3, pid: 27, ready: 2, running: 123, sleeping: 82, turnaround: 207
Priority 1, pid: 28, ready: 2, running: 130, sleeping: 82, turnaround: 214
Priority 2, pid: 29, ready: 2, running: 135, sleeping: 82, turnaround: 219
Priority 3, pid: 30, ready: 2, running: 137, sleeping: 82, turnaround: 221
Priority 1, pid: 31, ready: 2, running: 140, sleeping: 82, turnaround: 224
Priority 2, pid: 32, ready: 2, running: 146, sleeping: 82, turnaround: 230
Priority 3, pid: 33, ready: 2, running: 149, sleeping: 82, turnaround: 233
Priority 1, pid: 34, ready: 2, running: 153, sleeping: 82, turnaround: 237
Priority 2, pid: 35, ready: 2, running: 159, sleeping: 82, turnaround: 243
Priority 3, pid: 36, ready: 2, running: 161, sleeping: 82, turnaround: 245
Priority 1, pid: 37, ready: 2, running: 164, sleeping: 82, turnaround: 248
Priority 2, pid: 38, ready: 2, running: 169, sleeping: 82, turnaround: 253
Priority 3, pid: 39, ready: 2, running: 175, sleeping: 82, turnaround: 259
Priority 1, pid: 40, ready: 2, running: 180, sleeping: 82, turnaround: 264
Priority 2, pid: 41, ready: 2, running: 186, sleeping: 82, turnaround: 270
Priority 3, pid: 42, ready: 2, running: 191, sleeping: 82, turnaround: 275
Priority 1, pid: 43, ready: 2, running: 196, sleeping: 82, turnaround: 280
Priority 2, pid: 44, ready: 2, running: 200, sleeping: 82, turnaround: 284

Priority 1:
Average ready time: 2
Average running time: 137
Average sleeping time: 82
Average turnaround time: 221

Priority 2:
Average ready time: 2
Average running time: 142
Average sleeping time: 82
Average turnaround time: 226

Priority 3:
Average ready time: 2
Average running time: 134
Average sleeping time: 82
Average turnaround time: 218

$
```

How to use the patch file:

- Keep xv6 cloned repo and the submitted **patchfile.patch** in the same directory. (Preferably a new folder in desktop to avoid errors)
    - (Link to clone xv6: ***git clone git://github.com/mit-pdos/xv6-public.git***)
- Open this parent directory (that contains patch and xv6) in the terminal and run the following command-
    - ***patch -s -p0 < patchfile.patch***
- The xv6-public folder will now have all the required changes.