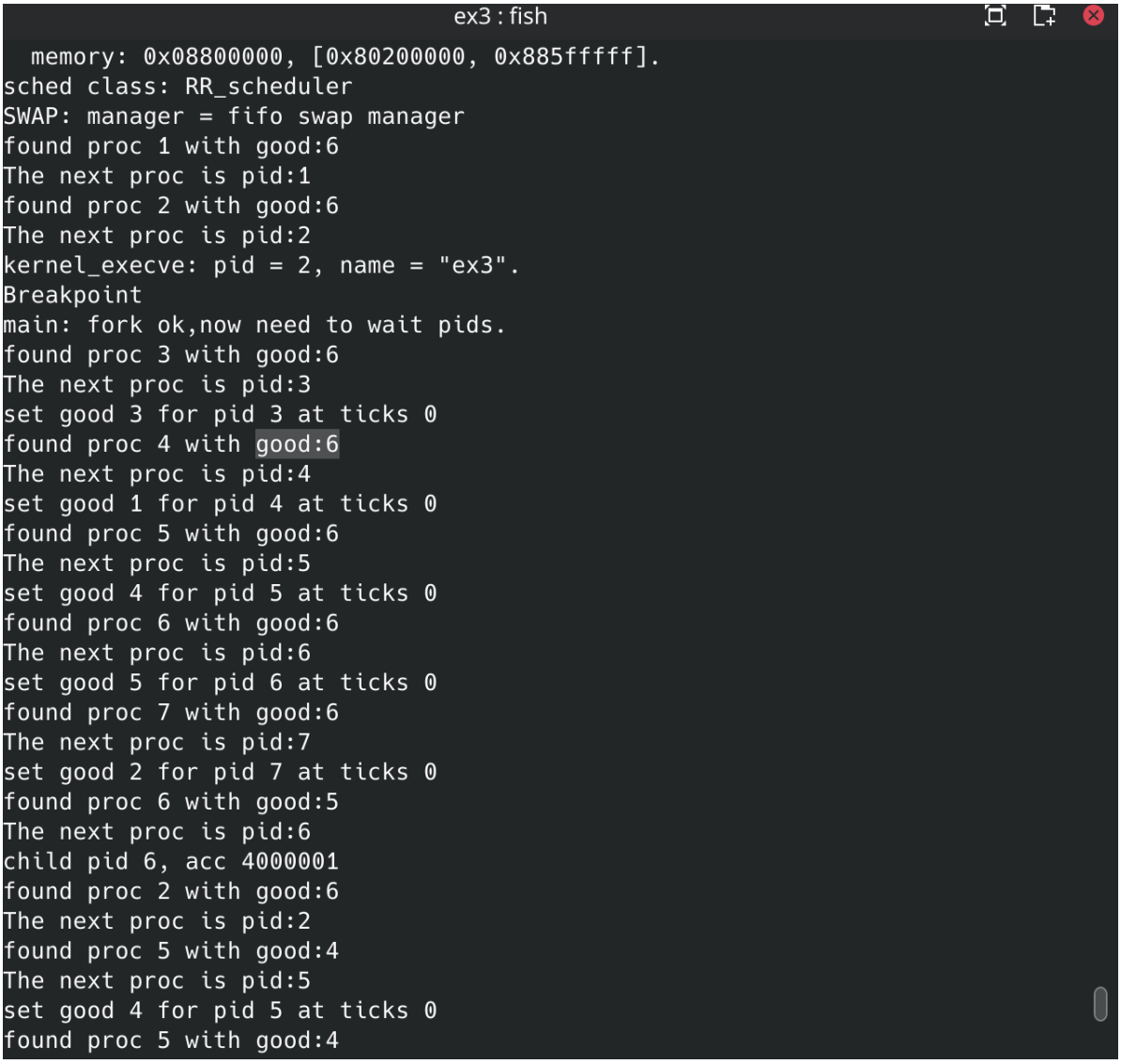
ass4

Q1:

Time	HRRN	FIFO/FCFS	RR	SJF	PRiority
1	A	A	A	A	A
2	A	A	A	A	B
3	A	A	B	A	A
4	A	A	A	A	D
5	B	B	D	B	D
6	D	D	A	D	C
7	D	D	C	D	C
8	C	C	D	C	C
9	C	C	C	C	A
10	C	C	C	C	A
Avg. Turn-around Time	4.5	4.5	4.25	4.5	4.75

Q2:

modify `RR_pick_next` to choose process with the max good value


ex3 : fish
memory: 0x08800000, [0x80200000, 0x885fffff].
sched class: RR_scheduler
SWAP: manager = fifo swap manager
found proc 1 with good:6
The next proc is pid:1
found proc 2 with good:6
The next proc is pid:2
kernel_execve: pid = 2, name = "ex3".
Breakpoint
main: fork ok,now need to wait pids.
found proc 3 with good:6
The next proc is pid:3
set good 3 for pid 3 at ticks 0
found proc 4 with good:6
The next proc is pid:4
set good 1 for pid 4 at ticks 0
found proc 5 with good:6
The next proc is pid:5
set good 4 for pid 5 at ticks 0
found proc 6 with good:6
The next proc is pid:6
set good 5 for pid 6 at ticks 0
found proc 7 with good:6
The next proc is pid:7
set good 2 for pid 7 at ticks 0
found proc 6 with good:5
The next proc is pid:6
child pid 6, acc 4000001
found proc 2 with good:6
The next proc is pid:2
found proc 5 with good:4
The next proc is pid:5
set good 4 for pid 5 at ticks 0
found proc 5 with good:4

```
found proc 5 with good:4
The next proc is pid:5
set good 4 for pid 5 at ticks 0
found proc 5 with good:4
child pid 5, acc 4000001
found proc 2 with good:6
The next proc is pid:2
found proc 3 with good:3
The next proc is pid:3
set good 3 for pid 3 at ticks 0
found proc 3 with good:3
child pid 3, acc 4000001
found proc 2 with good:6
The next proc is pid:2
found proc 7 with good:2
The next proc is pid:7
child pid 7, acc 4000001
found proc 2 with good:6
The next proc is pid:2
found proc 4 with good:1
The next proc is pid:4
child pid 4, acc 4000001
found proc 2 with good:6
The next proc is pid:2
main: wait pids over
found proc 1 with good:6
The next proc is pid:1
found proc 1 with good:6
all user-mode processes have quit.
The end of init_main
kernel panic at kern/process/proc.c:414:
    initproc exit.

~/g/o/a/ex3  master ±
```

init.c U | default_sched.c U | syscall.c .../syscall U | sched.c U | syscall.c .../libs U | syscall.h U | ex3.c U | ulib.h U | ulib.c U | unistd.h U

kern > init > init.c > kern_init(void)

```c
#include <trap.h>
#include <clock.h>
#include <intr.h>
#include <pmm.h>
#include <vmm.h>
#include <ide.h>
#include <swap.h>
#include <proc.h>
#include <kmonitor.h>

int kern_init(void) __attribute__((noreturn));
int
kern_init(void) {
    extern char edata[], end[];
    memset(edata, 0, end - edata);
    cons_init();                // init the console

    const char *message = "OS is loading ...";
    cprintf("%s\n\n", message);

    pmm_init();                 // init physical memory management

    idt_init();                 // init interrupt descriptor table

    vmm_init();                 // init virtual memory management
    sched_init();
    proc_init();                // init process table

    ide_init();                 // init ide devices
    swap_init();                // init swap

    // clock_init();               // init clock interrupt
    intr_enable();              // enable irq interrupt

    cpu_idle();                 // run idle process
}

```

init.c U | default_sched.c U | syscall.c .../syscall U | sched.c U | syscall.c .../libs U | syscall.h U | ex3.c U | ulib.h U | ulib.c U | unistd.h U

kern > schedule > default_sched.c > RR_dequeue(run_queue *, proc_struct *)

```
    list_add_before(&(rq->run_list), &(proc->run_link));
    if (proc->time_slice == 0 || proc->time_slice > rq->max_time_slice) {
        proc->time_slice = rq->max_time_slice;
    }
    proc->rq = rq;
    rq->proc_num ++;
}

static void
RR_dequeue(struct run_queue *rq, struct proc_struct *proc) {
    assert(!list_empty(&(proc->run_link)) && proc->rq == rq);
    list_del_init(&(proc->run_link));
    rq->proc_num --;
}

static struct proc_struct *
RR_pick_next(struct run_queue *rq) {
    const list_entry_t *start = &(rq->run_list);
    list_entry_t *max = NULL;
    list_entry_t *le = start->next;
    while (le && le != start) {
        if(!max){
            max = le;
        }
        else if(le2proc(le, run_link)->labschedule_good > le2proc(max, run_link)->labschedule_good){
            max = le;
        }
        le = list_next(le);
    }
    if(max){
        cprintf("found proc %d with good:%d\n", le2proc(max, run_link)->pid,le2proc(max, run_link)->labschedule_good);
    }else{
        return NULL;
    }
    return le2proc(max, run_link);
}

static void
RR_proc_tick(struct run_queue *rq, struct proc_struct *proc) {
    if (proc->time_slice > 0) {
        proc->time_slice --;
    }
    if (proc->time_slice == 0) {
        proc->need_resched = 1;
    }
}
```

init.c U | default_sched.c U | syscall.c .../syscall U | sched.c U | syscall.c .../libs U | syscall.h U | ex3.c U | ulib.h U | ulib.c U | unistd.h U

kern > syscall > syscall.c > sys_set_good(uint64_t [])

```
static int
sys_putc(uint64_t arg[]) {
    int c = (int)arg[0];
    cputchar(c);
    return 0;
}

static int sys_gettime(uint64_t arg[]){
    return (int)ticks*10;
}

static int sys_set_good(uint64_t arg[]){
    uint32_t good = (uint32_t)arg[0];
    assert(current!=NULL);
    current->labschedule_good = good;
    cprintf("set good %d for pid %d at ticks %d \n",good,current->pid,ticks);
    current->need_resched = 1;
    schedule();
    return 0;
}

static int (*syscalls[])(uint64_t arg[]) = {
    [SYS_exit]              sys_exit,
    [SYS_fork]              sys_fork,
    [SYS_wait]              sys_wait,
    [SYS_exec]              sys_exec,
    [SYS_yield]             sys_yield,
    [SYS_kill]              sys_kill,
    [SYS_getpid]            sys_getpid,
    [SYS_putc]              sys_putc,
    [SYS_gettime]           sys_gettime,
    [SYS_set_good]         sys_set_good,
};

#define NUM_SYSCALLS        ((sizeof(syscalls)) / (sizeof(syscalls[0])))

void
syscall(void) {
    struct trapframe *tf = current->tf;
```

init.c U | default_sched.c U | syscall.c .../syscall U | sched.c U | syscall.c .../libs U | syscall.h U | ex3.c U | ulib.h U | ulib.c U | unistd.h U

user > libs > syscall.c > sys_set_good(uint32_t)

```
int
sys_getpid(void) {
    return syscall(SYS_getpid);
}

int
sys_putc(int64_t c) {
    return syscall(SYS_putc, c);
}

int
sys_gettime(void) {
    return syscall(SYS_gettime);
}

int sys_set_good(uint32_t good){
    return syscall(SYS_set_good,good);
}
```

init.c U | default_sched.c U | syscall.c .../syscall U | sched.c U | syscall.c .../libs U | syscall.h U | ex3.c U | ulib.h U | ulib.c U | unistd.h U

user > libs > syscall.h > sys_set_good(uint32_t)

```c
#ifndef __USER_LIBS_SYSCALL_H__
#define __USER_LIBS_SYSCALL_H__

int sys_exit(int64_t error_code);
int sys_fork(void);
int sys_wait(int64_t pid, int *store);
int sys_yield(void);
int sys_kill(int64_t pid);
int sys_getpid(void);
int sys_putc(int64_t c);
int sys_gettime(void);
int sys_set_good(uint32_t good);


#endif /* !__USER_LIBS_SYSCALL_H__ */

```

init.c U | default_sched.c U | syscall.c .../syscall U | sched.c U | syscall.c .../libs U | syscall.h U | ex3.c U | ulib.h U | ulib.c U | unistd.h U

user > libs > ulib.h > set_good(uint32_t)

```c
#ifndef __USER_LIBS_ULIB_H__
#define __USER_LIBS_ULIB_H__

#include <defs.h>

void __warn(const char *file, int line, const char *fmt, ...);
void __noreturn __panic(const char *file, int line, const char *fmt, ...);

#define warn(...)                                       \
    __warn(__FILE__, __LINE__, __VA_ARGS__)

#define panic(...)                                      \
    __panic(__FILE__, __LINE__, __VA_ARGS__)

#define assert(x)                                       \
    do {                                                \
        if (!(x)) {                                     \
            panic("assertion failed: %s", #x);          \
        }                                               \
    } while (0)

// static_assert(x) will generate a compile-time error if 'x' is false.
#define static_assert(x)                                \
    switch (x) { case 0: case (x): ; }

void __noreturn exit(int error_code);
int fork(void);
int wait(void);
int waitpid(int pid, int *store);
void yield(void);
int kill(int pid);
int getpid(void);
unsigned int gettime_msec(void);
void set_good(uint32_t good);

#endif /* !__USER_LIBS_ULIB_H__ */

```

init.c U | default_sched.c U | syscall.c .../syscall U | sched.c U | syscall.c .../libs U | syscall.h U | ex3.c U | ulib.h U | ulib.c U | unistd.h U

user > libs > ulib.c > gettime_msec(void)

```c
}

int
getpid(void) {
    return sys_getpid();
}

unsigned int
gettime_msec(void) {
    return (unsigned int)sys_gettime();
}

void set_good(uint32_t good){
    sys_set_good(good);
}

```

init.c U | default_sched.c U | syscall.c .../syscall U | sched.c U | syscall.c .../libs U | syscall.h U | ex3.c U | ulib.h U | ulib.c U | unistd.h U

libs > unistd.h > SYS_set_good

```c
#ifndef __LIBS_UNISTD_H__
#define __LIBS_UNISTD_H__

#define T_SYSCALL           0x80

/* syscall number */
#define SYS_exit            1
#define SYS_fork            2
#define SYS_wait            3
#define SYS_exec            4
#define SYS_clone           5
#define SYS_yield           10
#define SYS_sleep           11
#define SYS_kill            12
#define SYS_gettime         17
#define SYS_getpid          18
#define SYS_brk             19
#define SYS_mmap            20
#define SYS_munmap          21
#define SYS_shmem           22
#define SYS_putc            30
#define SYS_pgdir           31
#define SYS_set_good         254
/*only for labschedule*/
#define SYS_labschedule_set_priority 255

/* SYS_fork flags */
#define CLONE_VM            0x00000100  // set if VM shared between processes
#define CLONE_THREAD        0x00000200  // thread group

#endif /* !__LIBS_UNISTD_H__ */

```