NORTHEASTERN UNIVERSITY
COLLEGE OF ENGINEERING
Department of Electrical & Computer Engineering

96

EECE7376: Operating Systems: Interface and Implementation — **Dr. Emad Aboelela**
Spring 2024 – Quiz 3 — **Time Allowed: 1.5 Hour**

Student Name: _____ Student ID: _____

This is a **closed book** and **closed notes** quiz. The quiz has **7 questions**
You **are not allowed** to use any **electronic** device nor **scratch** papers
No answers outside these quiz pages will be graded (you can write on **both sides**)
Make sure to **write your full name** on the empty side of your cheat sheet
You cannot leave the room once the quiz starts (except in case of an emergency)

Q1. (28 Points)

28 Answer each of the following questions in one sentence:

a) The following `if` statement appears in the code of the xv6 `syscall()` function:

```
if(num > 0 && num < NELEM(syscalls) && syscalls[num])
```

where `num` is assigned the value `curproc->tf->eax`

Why does the function need to check that `syscalls[num]` is true?

b) Why is a lock needed as part of the struct that defines the process table `ptable` (in `proc.c`)?

c) xv6 maps the entire code of its kernel into the virtual memory image of each process. What is the advantage of doing this?

d) In the virtual address space of an xv6 process, why is the stack address space follwed by a guard page that is not mapped to any physical address space?

Q2. (14 Points)

10 In xv6, the system call `sleep(n)` is used to put the process to sleep for *n* seconds and its system call number is defined in `syscall.h` as `#define SYS_sleep 13`

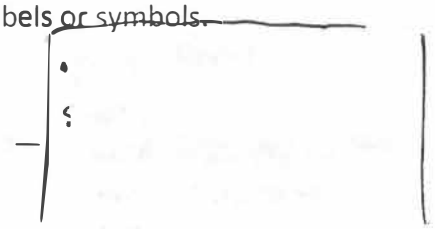
The following macro in `usys.S` is used to generate the x86 code corresponding to any system call:

```
#include "syscall.h"
#include "traps.h"

#define SYSCALL(name) \
  .globl name; \
  name: \
    movl $SYS_ ## name, %eax; \
    int $T_SYSCALL; \
    ret
```

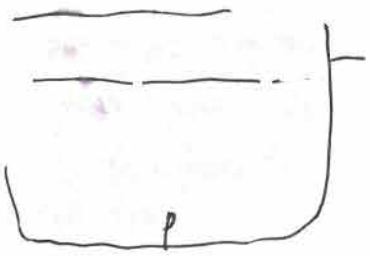
Answer the following questions:

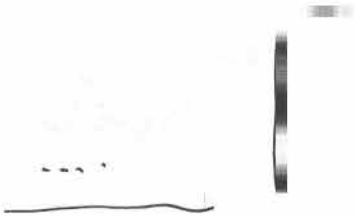
a) If a process calls `sleep(5)` to sleep for 5 seconds and based on the above macro, what will -1 be the three x86 machine code instructions corresponding to `SYSCALL(sleep)`? *Note:* machine code means instructions without assembly labels or symbols.

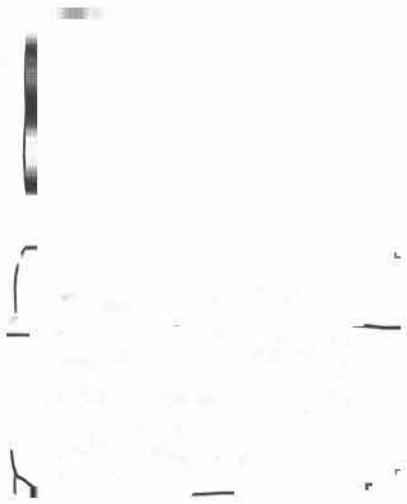


b) Calling `sleep(5)` requires passing two numbers to the multiple code functions that are needed to complete the system call execution. These two numbers are 5 and 13. Explain how -3 and where these two numbers are passed to those functions.

You need to specifically say how a pa







Q3. (10 Points)

10

Assume that `malloc()` always returns successfully. Select (using a check mark) **ALL** possible outputs of the shown code from the following list:

- ☐ 12345 ✗
- ☑ 01234
- ☑ 03142
- ☐ 01122 ✗
- ☐ 44444 ✗

Points will be deducted if you make a wrong selection or if you miss a correct possible output.

```
void* printer(void* arg) {
      char* p = (char*)arg;
      printf("%d", *p);
      free(p);
      return NULL;
}

int main(int argc, char* argv[]) {
  pthread_t p[5];
  for (char i = 0; i < 5; i++) {
    char* c = malloc(sizeof(char));
    *c = i;
    pthread_create(&p[i], NULL, printer, (void*)c);
  }

  for (char i = 0; i < 5; i++)
      pthread_join(p[i], NULL);

 return 0;
}
```

Q4. (10 Points)

10

Select (using a check mark) **ALL** possible outputs of the shown code from the following list:

- ☑ 01234 ✓
- ☑ 22344 ✓
- ☐ 22224 ✗
- ☑ 43210 ✓
- ☑ 44444 ✓
- ☐ 00000 ✗

Points will be deducted if you make a wrong selection or if you miss a correct possible output.

```
void* printer(void* arg) {
      char* p = (char*)arg;
      printf("%d", *p);
      return NULL;
}

int main(int argc, char* argv[]) {
  pthread_t p[5];
  char c;

  for (char i = 0; i < 5; i++) {
    c = i;
    pthread_create(&p[i], NULL, printer, (void*)&c);
  }

  for (char i = 0; i < 5; i++)
      pthread_join(p[i], NULL);

  return 0;
}
```

Q5. (10 Points)

10

Select (using a check mark) **ALL** possible outputs of the shown code from the following list:

- ☐ 0
- ☑ 98
- ☑ 99
- ☐ 100

Points will be deducted if you make a wrong selection or if you miss a correct possible output.

```
int counter = 100;

void* decrement(void* arg) {
      counter--;
      return NULL;
}

int main(int argc, char* argv[]) {
  pthread_t p1, p2;

  pthread_create(&p1, NULL, decrement, NULL);
  pthread_create(&p2, NULL, decrement, NULL);

  pthread_join(p1, NULL);
  pthread_join(p2, NULL);

  printf("%d\n", counter);

  return 0;
}
```

Q6. (10 Points)

10

Assume that `malloc()` always returns successfully and that a not shown function to free the allocated memory is called before the program exits. Select (using a check mark) **ALL** possible outputs of the shown code from the following list:

- ☐ #
- 1#
- ☑ 2#
- ☑ 21#
- ☑ 12#

Points will be deducted if you make a wrong selection or if you miss a correct possible output.

```
typedef struct node_t {
      int value;
      struct node_t* next;
};

struct node_t* List_Head = NULL;

int List_Insert(void* arg) {
  int* k = (int*)arg;
  struct node_t* n = malloc(sizeof(struct node_t));
  n->value = *k;
  n->next = List_Head;
  List_Head = n;
  return 0;
}

void PrintList() {
  struct node_t* p = List_Head;

  while (p != NULL) {
      printf("%d", p->value);
      p = p->next;
  }
  printf("#\n");
  return;
}

int main(int argc, char* argv[]) {
  pthread_t p1, p2;
  int a = 1, b = 2;

  pthread_create(&p1, NULL, List_Insert, (void*)&a);
  pthread_create(&p2, NULL, List_Insert, (void*)&b);

  pthread_join(p1, NULL);
  pthread_join(p2, NULL);

  PrintList();

  return 0;
}
```

Q7. (18 Points)

18

The following three lines of x86 assembly code correspond to the increment of an integer counter residing in memory address `0x804a02c`

```
load:    mov 0x804a02c, %eax
add:     add $0x1, %eax
store:   mov %eax, 0x804a02c
```

Write a timeline that illustrates the occurrence of a race condition between two threads, *A* and *B*, each running these three lines of code. Your timeline must highlight the situation where the counter will end up with a wrong final value. Assume that the initial value of the counter is zero.

Your timeline should include the following columns:

i) current instruction run by thread *A* (load, add, or store),

ii) current instruction run by thread *B* (load, add, or store)

iii) current value of register `eax` for each thread,

iv) current value of the shared counter in memory.

In the timeline, specify the times when you decide to introduce context switches that will cause that final wrong counter value.

End of the Quiz Questions