# UNIVERSITY OF LONDON

# BSc EXAMINATION 2023

For Internal Students of
Royal Holloway

## DO NOT TURN OVER UNTIL TOLD TO BEGIN

## CS2850: Operating Systems
## CS2850R: Operating Systems – for FIRSTSIT/RESIT CANDIDATES

Time Allowed: **TWO hours**

Please answer **ALL** questions

1. **Theory Questions (25 marks).**

   (a) In the context of memory management, what is the difference between "static" and "dynamic" relocation? [3 marks]

   (b) Briefly describe the linked-list allocation method for storing files. Give one advantage of this method over the contiguous allocation one. [4 marks]

   (c) In a mainframe system without multiprogramming (i.e., programs cannot be run concurrently), consider four jobs A,B, C, and D. Suppose that their estimated running times are:

| | running time (minutes) |
|---|---|
| A | 4 |
| B | 20 |
| C | 3 |
| D | 18 |

   Give the **turnaround times** for each of these jobs and the **average turnaround time** in the context of the following two batch system scheduling strategies, making sure to show all your working.

   i. First-Come First-Served (FCFS), assuming the order A, B, C, D. [5 marks]

   ii. Shortest Job First (SJF). [5 marks]

   (d) Consider the program that consists of the following two concurrent processes that share the variable `x`:

```
P1:   x = 4;
      x = x + 2;
```

```
P2:   x = 1;
      if (x == 4) {
        x = x * 5;
      }
      else {
        x = x - 1;
      }
```

   How many different values may the variable `x` have when the program terminates? Give a possible execution sequence for each case. [8 marks]



**NEXT PAGE**

2. **Theory Questions (25 marks).**

(a) Consider the following proposed solution to the mutual exclusion problem for two processes. Suppose that the two processes share a variable `v`, initially set to 0.

```
P1:
while(1) {
  if (v % 2 == 0) {
    critical_region1();
    v++;
    non_critical_region1();
  }
}
```

```
P2:
while(1) {
  if (v % 2 == 1) {
    critical_region2();
    v++;
    non_critical_region2();
  }
}
```

i. Briefly describe the operations of the algorithm, giving particular emphasis to any dependencies between the two processes. Explain why the algorithm does enforce strict alternation. [6 marks]

ii. This algorithm does not comply with one of the conditions required for solving the mutual exclusion problem. Indicate which one, and give one example of execution order violating that condition. [5 marks]

(b) Briefly describe the Banker's algorithm with a single resource type, and indicate how it can be used by the Operating System to determine whether a resource request made by a process should be granted. [6 marks]

(c) A virtual memory system has 32-bit virtual addresses and page size 4K. To optimise memory usage, it uses a two-level page table, where the left-most 12 bits of the virtual address are used to index the top-level table.

i. How many second-level page tables are there in total? [4 marks]

ii. How many entries does each second-level page table have? [4 marks]

Justify your answers.

3. **Pointers (20 marks)**.

```
#include <stdio.h>

struct point {
 int x;
 int y;
 char label;
};

int main() {
 int i = 1.234;
 printf("1. (float i) = %f\n", (float) i);
 printf("2. (void *) &i = %p\n", (void *) &i);
 int j = 12.345;
 printf("3. j = %o\n", j);
 int *ip = &i;
 int *jp = &j;
 printf("4. (void *) (&i - jp) = %p\n", (void *) (&i - jp));
 printf("5. *(&i + 100) = %d\n", *(&i + 100));
 ip = &j;
 *ip = i;
 printf("6. i - j = %d\n", i - j);
 printf("7. sizeof(struct point) = %lu\n", sizeof(struct
   point));
 struct point z;
 printf("8. sizeof(&z) = %lu\n", sizeof(&z));
 (&z)->x = i;
 z.y = j;
 (&z)->label = '1';
 printf("9. (&z)->label = %d\n", (&z)->label);
 printf("10. z.label = %c\n", z.label);
}
```

A possible `stdout` output of the program above is

```
1. (float i) = 1.000000
2. (void *) &i = 0x7ffe7c124520
3. j = 14
4. (void *) (&i - jp) = 0xffffffffffffffff
5. *(&i + 100) = 2081580231
6. i - j = 0
7. sizeof(struct point) = 12
8. sizeof(&z) = 8
9. (&z)->label = 49
10. z.label = 1
```

For each line of the output, explain what is printed on the terminal and why you obtain those specific values. [20 marks]

4. **Swapping functions (10 marks).**

The output of the program in Listing 1 when the user enters the character `q` to relabel the point called `'1'` is

```
point 1: (x = 1, y = 2)
point 2: (x = 3, y = 4)
swapping x coordinates ...
point 1: (x = 3, y = 2)
point 2: (x = 1, y = 4)
enter a new label for point 1:
q
the new label for point 1 is q
point q: (x = 3, y = 2)
point 2: (x = 1, y = 4)
```

Write the definition of

(a) `swapx`, a function that swaps the `x` coordinate of the points, and [5 marks]

(b) `changeLabel`, a function that renames one point by asking the user to enter a character in the terminal, [5 marks]

so that the program in Listing 1 behaves exactly as in the output shown above.

**Hint:** In `changeLabel`, use `char c = getchar();` to read the user input from the terminal and store it in a character variable.

Listing 1: C code of the program

```
#include <stdio.h>
struct point {
 int x;
 int y;
 char label;
};
void swapx(int *p1, int *p2);
void changeLabel(struct point *pz);
void printPoints(struct point *p, struct point *q);
int main() {
 struct point z1, z2;
 (&z1)->x = 1;
 (&z1)->y = 2;
 (&z2)->x = 3;
 (&z2)->y = 4;
 (&z1)->label = '1';
 (&z2)->label = '2';
 printPoints(&z1, &z2);
 swapx(&(z1.x), &(z2.x));
 printPoints(&z1, &z2);
 changeLabel(&z1);
 printPoints(&z1, &z2);
}
void printPoints(struct point *p, struct point *q) {
 printf("point %c: (x = %d, y = %d)\n", p->label, p
   ->x, p->y);
 printf("point %c: (x = %d, y = %d)\n", q->label, q
   ->x, q->y);
}
```

5. **String merger (20 marks).**

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
int stringLen(char *string);
char *merge(char *string, char *s);
int main() {
 char c = '\0';
 char *string = NULL;
 while (c != EOF) {
  char s[MAX];
  int i = 0;
  while (((c = getchar())!='\n') && (i < (MAX - 1)) && (c !=
   EOF))
   *(s + i++) = c;
  *(s + i) = '\0';
  string = merge(string, s);
  printf("%s\n", string);
 }
 free(string);
}
```

The program above uses dynamic memory allocation to for handle a string of variable length. Read the definition of `stringLen` and `merge` in Listing 2 and answer the following questions.

(a) In `main`:

i. What does the program do and why do you need to allocate the memory dynamically? [4 marks]

ii. Why would `char c;` instead of `char c='\0'` produce an execution error? [2 marks]

iii. Why do you need `i< (MAX - 1)` instead of `i <MAX`? [2 marks]

iv. What happens if the user enters more than `MAX-1` characters? [2 marks]

v. Why do you need to null-terminate `s` before passing it to `merge`? [2 marks]

vi. Why do you need `free(string)` at the end? [2 marks]

(b) In `merge`:

i. What is the purpose of the first and the second `while` loops? [4 marks]

ii. Why do you need to check if `string` is a valid pointer before freeing it? [2 marks]

Listing 2: Definition of `stringLen` and `merge`

```
int stringLen(char *string) {
 if (!string) return 0;
 int i = 0;
 while (*(string + i) != '\0') i++;
 return i;
}

char *merge(char *string, char *s) {
 int n = stringLen(string);
 int m = stringLen(s);
 int i = 0;
 char *new = malloc(n + m + 1);
 while (i < n) {
  *(new + i) = *(string + i);
  i++;
 }
 while (i < (n + m)) {
  *(new + i) = *(s + i - n);
  i++;
 }
 *(new + i) = '\0';
 if (string) free(string);
 return new;
}
```

**END**