System and device programming
17 July 2017

(Theory: no textbooks and/or course material allowed)

15 marks. The final mark is the sum of the 1st and the 2nd part.

UNIX

To speed-up the correction process, please write your solutions for the UNIX questions on a sheet, and your solutions for the WINDOWS questions on a different sheet.

1. (3.0 marks) Write the pseudo-code of the Producers&Consumers problem **with conditions and locks** (not semaphores).

2. (a)

   Why many Memory Management Units use more than one field of the virtual address for paging

…	…	…	offset

   (b) What is the content of each field?

   (c) Draw the data structure that uses **four** fields and allows mapping a virtual address to the corresponding physical address.

   (d) Which is the overhead for this MMU?

   (e) What is the content of the fields of the Translation Lookaside Buffer for this MMU?

System and device programming
17 July 2017

WINDOWS

To speed-up the correction process, please write your solutions for the UNIX questions on a sheet, and your solutions for the WINDOWS questions on a different sheet.

3. (3.0 marks) Explain the role of filter expressions in try…except blocks. Why/when do the following block call the Filter functions ?

```
_try {
   _try {
     _try {


                                 [illegible] [illegible] ,/)) {
        printf("In except 1\n");
      }
    _finally {
      printf("In finally\n");
    }
} _except (Filter2 (GetExceptionCode ())) {
    printf("In except 2\n");
}
```

Are `Filter1` and `Filter2` system routines or user-generated functions ? What is the role of `GetExceptionCode()` ? Under which conditions is `Filter2` called ? Is `Filter2` called when `Filter1` returns EXCEPTION_ [illegible] EXECUTION ? *(motivate all yes/no answers),*

When is `"In finally\n"` printed?

Is it possible for the program to print messages in one (or more) of the following orders (motivate answers) ?

A	B	C	C
In except 1 In finally In except 2	In except 1 In except 2 In finally	In except 1 In finally	In except 2 In finally

Continues on the back side

4. (2.0 marks) Describe why the C library is not thread safe, and outline the main difference between the _beginthreadex and the CreateThread functions. Show why/when a program using CreateThread could be erroneous ?

Is a DLL also required to be thread safe? (motivate the answer)?

Given a DLL with dynamic "explicit" linking, do we need to call `LoadLibrary()` before every call to a library function? And `GetProcAddress()`? (motivate answers)

5. (4.0 marks) Explain the role of completion fucntions in extended (alertable) asynchronous IO.

A given file has to be created and filled with a sequence of nr fixed length records, that can be described by the following C struct:

```
typedef struct {
  data_t value;
  key_t key;
} record_t;
```

Types `data_t` and `key_t` are given (other struct). Records are numbered from 0 to nr-1 and contained in a given array R. Function `storeDataToFile` is in charge of storing data, given the file handle and the array. Write the missing part of `storeDataToFile` and of the completion routine `writeRecords` in order to achieve the task by means of Extended asynchronous IO using completion routines (use of global variables is allowed).

```
DWORD storeDataToFile(HANDLE hFile, record_t *R, DWORD nr) {

  ...

  WriteFileEx (hFile, …, readDone);

  ...

}

VOID WINAPI writeDone(DWORD Code, DWORD nBytes, LPOVERLAPPED
pOv) {

  ...

  ReadFileEx (hFile, …, writeDone);

  ...

}
```

System and device programming
17 July 2017

Examination Time: 1h 45min. Evaluation. 18 marks.

Textbooks and/or course material allowed.

The final mark is the sum of the 1st and the 2nd parts

The final mark cannot be refused if this is you second chance,

The final mark will be registered (no retry for marks >= 18)

Implement a program in concurrent C language using Pthreads, semaphores, pipes, and shared buffers, that simulates a **circular** underground line including **4** stations and **2** trains running in the same direction.

A train travel from a station to the next one in **T_TR** seconds, and stops at each station the time necessary to load the passengers that are waiting (up to **TRAIN_CAPACITY**) **or no more than T_MAX seconds from the time the last passenger got on the train**. Then it leaves for the next station.

If more than **TRAIN_CAPACITY** passengers are waiting on that station, the remaining ones must wait and take the next train.

The program must create:

- a thread for each station,
- a thread for each train, and,
- looping **L** times, a new passenger thread at random intervals (`0-T_NEW_PASS`).

Each **passenger thread** randomly decides its destination station, and its starting station, where it queues waiting that a train reaches that station.

When a passenger reaches its destination, the thread terminates.

A **train thread k** is created in a random station **`train[k].station`**. While a train is in a station, a "red" semaphore (associated to the station, but managed by the train at that station) does not allow another train currently in the previous station to proceed to this station.

Before leaving the station, the train thread:

- appends two lines to file **`log`**:
    - the first one includes the train identifier, the train current station, and the identifier of the passengers in the train,
    - the second line includes the destination station of the passengers.

- communicates to the station thread its identifier (`0 or 1`), and the total number of passengers in the train, and the number of passengers that reached their destination at that station.

The **station thread** waits the train information, and then prints its identifier (`0-3`), and the information received by the train (train identifier, and the number of passengers that are in the train or reached their destination at that station.

Use the appropriate data structures to keep the necessary information, **shared buffers** to manage the station queues, and a **pipe** to communicate information between a train and the station.

See an example of run on the back.

Example of output running `./trains-passengers` with `srand(1234)` and `T_TR 12 TRAIN_CAPACITY 10, T_MAX 3, T_NEW_PASS 10, L 1000`

```
Station 3: Train 1 passengers in: 1 passengers out: 0
Station 2: Train 0 passengers in: 0 passengers out: 0
Station 3: Train 0 passengers in: 2 passengers out: 0
Station 0: Train 1 passengers in: 2 passengers out: 0
Station 0: Train 0 passengers in: 4 passengers out: 0
Station 1: Train 1 passengers in: 1 passengers out: 2
Station 1: Train 0 passengers in: 4 passengers out: 1
Station 2: Train 1 passengers in: 4 passengers out: 1
Station 2: Train 0 passengers in: 2 passengers out: 3
Station 3: Train 1 passengers in: 5 passengers out: 2
```

Corresponding `log` file:

```
Train 1 Station 3    1
Destinations:        1
-----------------------------------------
Train 0 Station 2
Destinations:
-----------------------------------------
Train 1 Station 0    1   6
Destinations:        1   1
-----------------------------------------
Train 0 Station 3    2   5
Destinations:        2   2
-----------------------------------------
Train 1 Station 1    3
Destinations:        2
-----------------------------------------
Train 0 Station 0    9   2   5   8
Destinations:        1   2   2   2
-----------------------------------------
Train 1 Station 2    4  13  10  12
Destinations:        1   1   3   3
-----------------------------------------
Train 0 Station 1   11   2   5   8
Destinations:        0   2   2   2
-----------------------------------------
Train 1 Station 3    7  16   4  13  18
Destinations:        0   0   1   1   1
-----------------------------------------
Train 0 Station 2   11  17
Destinations:        0   1
-----------------------------------------
```

Usage template of `timedwait` system call:

```
ts.tv_sec = time (NULL) + atoi(argv[2]);
ts.tv_nsec = 0;
s = sem_timedwait(&sem, &ts);
if (s == -1) {
  if (errno == ETIMEDOUT)
    printf("timeut: sem counter not changed \n");
  else
    perror("sem_timedwait");
} else
   printf("no timeout: sem counter decremented \n");
```