Goals

In this session, you will begin to practice complex techniques typical of operating systems. This session will focus on the creation of threads in order to use several computer cores and thus carry out different tasks simultaneously. Likewise, basic forms of thread synchronization and the passing of variables between them will be practiced. Different thread settings should also be used.

Motivation

More specifically, with this session, the student must exercise:

- Threads creation (*pthread_create*).
- Passing parameters to threads.
- Threads synchronization (*pthread_join*).
- Attribute configuration (*pthread_attr_t*).

Previous documentation

To complete this session, it is recommended to read the following references:

SALVADOR, J. (2014). *Programació en UNIX per a pràctiques de Sistemes Operatius*. Publicacions d'Enginyeria i Arquitectura La Salle (Edición PDF). Pages 59 and 84.

STEVENS, W. R. & RAGO S.A. (2008). *Advanced Programming in the UNIX Environment*, 2nd edition.

TANENBAUM, A. S. (2009). *Modern Operating Systems*, 3rd Edition.

LS Triage: LifeSaving Triage

In the aftermath of a tragic building collapse at the construction site of La Salle Barcelona's new IASLAB building, the nearby hospital finds itself flooded with a surge of injured patients.

La Salle students from various disciplines join forces to design a cutting-edge triage system and software to efficiently categorize patients based on the severity of their injuries. As time ticks away, the collaborative effort between computer science, engineering, and medical students becomes a race against time, holding the potential to revolutionize emergency response capabilities and save lives in moments of crisis.

The information for each of the patient will be contained by a text file received as an argument. This file will have the following structure:

<N_PATIENTS>\n

<name>#<age>#<height>#<weight>#<open wounds (T/F)>#<conscious (T/F)>#<respiratory track(OK/KO)>#<cronic condition(T/F)>#<thoracoabdominal bleeding(T/F)>#<other injuries(T/F)>\n

<name>#<age>#<height>#<weight>#<open wounds (T/F)>#<conscious (T/F)>#<respiratory track(OK/KO)>#<cronic condition(T/F)>#<thoracoabdominal bleeding(T/F)>#<other injuries(T/F)>\n

<name>#<age>#<height>#<weight>#<open wounds (T/F)>#<conscious (T/F)>#<respiratory track(OK/KO)>#<cronic condition(T/F)>#<thoracoabdominal bleeding(T/F)>#<other injuries(T/F)>\n

….

Each of these fields will help us in our task. We need to perform a triage, and assign a priority from 0 to 3 to all the patients (3 being the most critical, and 0 the least). This priority will help us order the medical care, and ensure those in critical condition are attended first. These priorities need to be assigned simultaneously, and as quick as possible. We want all the passengers file to be processed in parallel, and whenever each patient has been assigned a priority it will then be sent to the "waiting room" to await instructions.

This translates to the process having to wait for all the triage to finish before starting to cure anyone. This is because, we want to ensure we don't start curing low priorities, and attend first everyone of high priority.

The criteria to assign a priority is:

- **Priority 3:** Is given to patients with **respiratory track KO** and **unconscious** (one implies the other).
- **Priority 2:** Patients with **respiratory track OK** but **unconscious** or to patients who are **conscious** but with active **thoracoabdominal bleeding**.
- **Priority 1:** Is the priority assigned to patients who are **conscious** with **other injuries** and **respiratory track OK** or to patients with **chronic condition**.
- **Priority 0:** The rest.

Once the triage is complete, patients will be sent into urgent care in groups of 5, of course in order of priority. Each group of 5 will wait for all the members of the group to be cured before sending in the next group. Essentially, patients go into urgent care 5 by 5, until there are no patients left.

The 5 patients should be handled at the same time, by 5 different doctors. Each doctor will perform the necessary treatments. Since we are only simulating this behavior, and aren't actual doctors, we will simulate the treatment itself by priorities as well. This simulation will consist on waiting a certain number of seconds, each second equates to an hour in real life. The number of seconds depends on the priority assigned:

- **Priority 3:** The time a doctor takes to cure a priority 3 patient comes defined by the following equation: $1.5 + (7/2000)*((age-50)^2)$
- **Priority 2:** Patients in this priority will take 2s to cure. If their IMC is above or equal to 25, then 1.3s are added to the time, if the IMC is under 18.5, 1.5 should be added. To calculate the IMC, remember the formula is: $weight / (height^2)$
- **Priority 1:** Patients in priority 1 will take 2s as well, but if they have a chronic condition then it will take an additional 2 more seconds.
- **Priority 0:** All patients in this priority take 1s to be cured.

Once all patients have been cured, the execution of the program will finish.

Execution Examples

As you can see in the screenshot, first all the priorities are assigned. Then patients are treated in groups of 5.

Considerations

- This exercise is **NOT MEDICALLY ACCURATE** and intended as a simple threads exercise.
- The output of the program must be completely identical to what is shown in the execution examples. The treatment order may vary, but must always remain in order of priority.
- The use of global variables is prohibited.
- SIGINT must be ignored during the entire execution. Program must not stop if one is received.
- The text file will have the specified format and there will be no mistakes.
- You must ensure that, when the execution ends, all the processes have ended properly.
- When the program ends, all the dynamic memory used must have been freed, and all file descriptors must be closed.
- All inputs and outputs must be done using file descriptors. **The use of printf, scanf, FILE*, getchar or similar is NOT allowed.**
- **The use of the functions "system", "popen", or from the same family is NOT allowed**.
- **You must compile using the –Wall, –Wextra, and –lpthread flags.**
- **Any deliverable containing warnings or errors will be directly discarded.**
- At the start of the .c you must include a comment with your logins, names, and surnames.
- For submitting the session, you must hand over a file "S3.c", and deliver it through eStudy platform.
- For the use of certain functions, it might be necessary to add:

```
#define _XOPEN_SOURCE 500

#define _POSIX_C_SOURCE 1
```