NF05 – INTRODUCTION AU LANGAGE C

Compte-rendu de projet

Thomas Girod – Kevin Hernandez

Automne

utt
UNIVERSITÉ DE TECHNOLOGIE
TROYES

Table des matières

Introduction

Le présent document est notre rapport de projet de NF05.

Dans le cadre de ce projet, nous avons dû simuler le fonctionnement du service des urgences d'un hôpital, représenté par le schéma suivant :

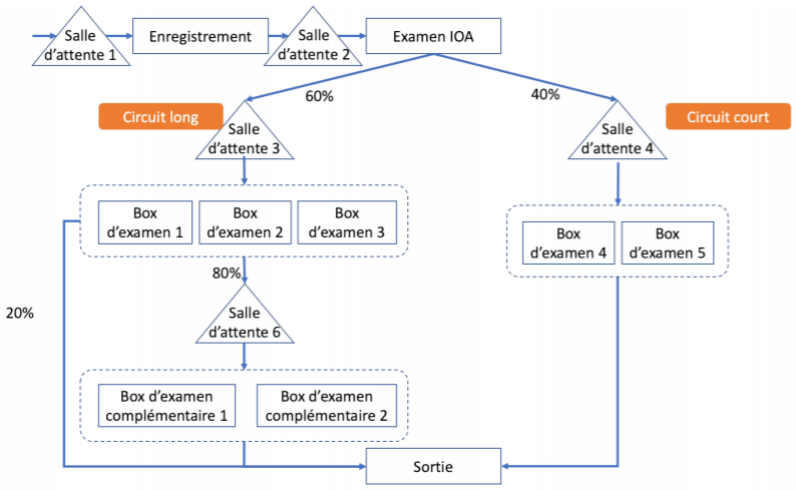

FIGURE 1 – Un flux simplifié d'un service d'urgences

Tous les patients rentrent dans ce circuit en salle d'attente 1 et en sortent par la sortie du service. Chaque nouveau patient doit se faire enregistrer en salle d'enregistrement, puis il passe devant l'infirmière organisatrice d'accueil (IOA). Une fois examiné, le patient est orienté soit vers le circuit court, soit vers le circuit long. Pour simplifier, on considère que 40% des patients empruntent le circuit court, et 60% le circuit long.

Le circuit court est dédié aux patients légers, ne nécessitant qu'un seul examen. Le circuit long est pour les patients plus graves ; selon leur état, les patients de ce circuit peuvent soit sortir après un premier examen (20% des cas), soit devoir passer un second examen avant de pouvoir sortir (80% des cas). Avant l'enregistrement et avant chaque examen, le patient doit attendre qu'une salle d'examen soit libre. Une fois que le patient précédent libère la salle, le patient suivant peut rentrer. L'entrée en salle d'examen se fait par ordre d'arrivée dans la salle d'attente qui précède la salle d'examen.

Toutes les données sur les patients (heure d'arrivée et durée de chaque examen) sont données avant le lancement de la simulation. L'entrée de ces données peut se faire soit manuellement par l'utilisateur soit être généré pseudo-aléatoirement par l'ordinateur.
Dans tous les cas, le nombre de patients est donné par l'utilisateur, et le circuit emprunté l'utilisateur est généré aléatoirement. Une fois les données rentrées, la simulation se lance.

Lorsque la simulation est achevée, le programme affiche un résumé des informations sur chaque patient : temps d'attente, temps d'examen, heure d'arrivée, heure de sortie, etc. Le programme affiche également une moyenne du temps d'attente et d'examen moyen de tous les patients.

Pour le sujet de projet complet, voir ce lien.

Ce rapport décrit successivement :
- La démarche adoptée pour créer un programme à même d'effectuer la simulation de ce service d'urgence, sans expliquer le code en détail ;
- Le mode d'emploi du programme pour l'utilisateur ;
- Les problèmes rencontrés au cours de l'implémentation du programme.

Le code n'est pas décrit en détail, mais il peut être trouvé avec sa documentation via les liens fournis en annexe.

I. Description du programme

les fonctions et procédures utilisées au cours de l'exécution du programme sont réparties entre les modules suivants (en plus du main, dans lequel on trouve uniquement la fonction principale du programme) :

- *input_ patients*, comprenant les procédures modifiant l'état d'une instance patient sans toucher à une instance salle d'attente ou salle d'examen ;
- *room_ management*, comprenant les procédures modifiant l'état d'une instance salle d'attente et/ou salle d'examen, ainsi que les instances patient qui interagissent avec ces instances ;
- *sort*, comprenant les procédures pour ordonner des listes lors de l'appel de la fonction qsort().

Le programme comprend en outre le fichier *struct.h*, un header qui n'est pas associé à un fichier source en particulier et qui comprend la déclaration de toutes les structures et énumérations utilisées au cours de l'exécution.

1 Les structures

Les principales structures utilisées sont :

- *patient*, décrivant un patient par son nom, prénom, circuit emprunté, durée de chaque examen, heure de début et de fin de chaque, et à tout moment de sa présence dans l'hôpital, l'heure à partir de laquelle il peut passer un nouvel examen ;
- *waitingRoom*, décrivant une salle d'attente par le nombre de patients qui s'y trouve et par un tableau contenant ces patients ;
- *examRoom*, décrivant une salle d'examen par l'heure à laquelle la salle est disponible au plus tôt ;
- *box*, structure qui comprend plusieurs salles d'examen, pour simuler les groupes de salles d'examen dans les circuits d'examen.

Les autres structures et les énumérations sont là pour simplifier le code et améliorer sa lisibilité.

Il est à noter que dans le programme, la sortie est considérée comme une salle d'attente. On considère également l'existence d'une salle d'attente virtuelle, située entre les deux groupes de salles d'examen du circuit long.

La salle d'enregistrement et la salle d'examen IOA sont considérées comme des salles d'examen.

2 Les fonctions et procédures (hors main)

2.1 Les fonctions de gestion des patients seuls

La fonction la plus importante pour la gestion des patients est la fonction input_patient _informations() ; cette fonction est celle utilisée en début de programme pour saisir manuellement les informations sur un patient (si l'utilisateur a fait ce choix). Au cours de la fonction, les différentes informations sur le patient (nom, prénom, heure d'arrivée et durée des examens) sont demandées à l'utilisateur et l'instance patient est modifiée en conséquence.

Au cours de son exécution, cette fonction fait appel à une autre fonction qui génère aléatoirement le circuit emprunté par le patient et le renvoie à la fonction d'entrée des informations du patient.

Pour toutes les données dont l'utilisateur a demandé la génération pseudo-aléatoire (nom, durée des examens ou heure d'arrivée), la fonction input_patient_informations() appelle d'autres sous- fonctions pour la génération aléatoire des nombres. La durée des examens est générée selon la méthode de Box-Muller. L'heure d'arrivée est l'heure d'arrivée du patient précédent à laquelle on ajoute le temps qui sépare le patient actuel de ce dernier ; ce temps de différence est généré selon une loi exponentielle.

2.2 Les fonctions de gestion des salles

Deux fonctions sont utilisées en début de programme pour initialiser l'état des différentes salles :

- Une première fonction pour déclarer toutes les salles d'attente comme vides ;
- Une deuxième fonction pour fixer l'heure de disponibilité des salles d'examen (00h00 en début de programme).

L'intérêt de cette deuxième fonction réside en ce que si le patient est disponible avant la salle, il attendra que l'heure de libération de la salle arrive. En fixant cette heure à 00h00, on s'assure que quelque soit l'heure à laquelle le premier patient arrivera, il pourra rentrer.

Deux autres fonctions sont utilisées lors de la simulation pour modifier l'état des salles :

- Une première fonction pour faire passer l'examen : le patient quitte la salle d'attente, l'heure de libération de la salle d'examen dans laquelle il passe est mise à jour, puis le patient passe dans la salle suivante ; l'heure de début et l'heure de fin d'examen sont enregistrées dans l'instance patient correspondante.
- Une deuxième fonction qui rajoute explicitement le patient dans la liste des patients présents dans une salle d'attente. Cette fonction n'est pas incluse dans la première, puisqu'elle n'est pas appelée pour tous les examens, mais uniquement pour ceux postérieurs à la séparation entre les deux circuits. Elle n'est vraiment utile qu'après la bifurcation ; avant ça, son usage son usage n'est pas nécessaire, puisque tous les patients passent dans les salles pré-bifurcation et que leur ordre d'arrivée en salle ne change pas non plus.

3 Le main et le déroulement général du programme

La fonction main du problème gère l'appel des fonctions pour simuler le déroulement des examens pour tous les patients. Le programme se déroule dans l'ordre suivant :

1. Les salles d'attente sont déclarées comme vide ;

2. Les salles d'examen sont déclarées comme disponibles à partir de 00h00 ;
3. Sont demandés à l'utilisateur le nombre total de patients et les données qu'il veut rentrer manuellement ou générer automatiquement.
4. L'utilisateur rentre manuellement les informations sur les patients (sauf s'il a demandé leur génération aléatoire en début de programme). Le circuit emprunté est généré aléatoirement dans tous les cas. A ce moment, les patients sont placés en salle d'attente 1 ;
5. On fait passer l'examen d'enregistrement à tous les patients et on les fait tous passer dans la salle d'attente 2 ;
6. On fait passer l'examen IOA. Après l'examen, selon le circuit emprunté, on place le patient en salle d'attente 3 ou en salle d'attente 4 ;
7. On parcourt la liste des patients présents en salle d'attente 4 et on fait passer l'examen médical du circuit court aux patients qui ont été placés sur cette liste après l'examen IOA. Tous les patients sont redirigés vers la sortie des urgences après leur examen ;
8. On fait la même chose pour la salle d'attente 3, à la différence qu'une partie des patients peut sortir mais que les patients ayant besoin d'un examen complémentaire passent en salle d'attente 5 (la salle d'attente virtuelle située entre les deux groupes de salles d'examen du circuit long).
9. Enfin, on fait passer le dernier examen médical aux patients présents dans la cinquième salle d'attente. Ces patients peuvent ensuite sortir.

A ce moment, la simulation est terminée, et on a récupéré l'heure de début et de fin de tous les examens passés par chaque patient. On peut donc ouvrir l'interface graphique servant à présenter les résultats de la simulation.

Le fonctionnement de l'interface graphique n'est pas compliqué : on commence par afficher tout ce qui formera le contenu permanent de la fenêtre, à savoir presque tout, excepté les heures de passage propres à chaque patient.

II. Mode d'emploi pour l'utilisateur

Attention : Ce programme a été développé pour des ordinateurs 64-bits fonctionnant avec le système d'exploitation Windows-10. La portabilité du programme sur d'autres machines n'est pas assurée.

1 Entrée des informations

Après un message d'accueil, l'utilisateur commence par rentrer quelles données il veut que le programme génère aléatoirement, puis il saisit le nombre de patients passant la simulation :

```
Welcome in our hospital management program in C for NF05.
Please write in the shell the informations which you shall be asked for


Do you want the patient name to be randomly generated ? (y/n) : n
Do you want the exam duration to be randomly generated ? (y/n) : n
Do you want the arrival hour to be randomly generated ? (y/n) : n
Number of patients : 3
```

FIGURE 2 – Choix des données aléatoires et nombre de patients

Pour chaque question, si l'utilisateur rentre 'y', la donnée sera générée par le programme ; s'il rentre 'n', il devra la rentrer à la main. Si l'utilisateur répond 'y' aux trois questions, alors la saisie des données sur la console se termine et la simulation se lance puis le fenêtre de présentation des résultats s'ouvre. S'il répond 'n' à au moins une question, il devra saisir manuellement les données pour lesquelles il a fait ce choix pour tous les patients. Les données pouvant être générés par le programme sont :

- le nom et le prénom ;
- l'heure d'arrivée ;
- la durée des examens.

```
Enter new patient :

Surname : Jacques
Name : Dupond
Arrival hour (format : hh mm) : 10 14
registration duration (in minutes) : 11
Duration of the exam by the nurse reception organizer (in minutes) : 9
Long circuit.
Duration of the first medical exam : 34
This patient does need one more exam
Duration of the second medical exam : 28
```

FIGURE 3 – Saisie manuelle des données patient

L'heure doit être entrée sous le format « hh mm », avec un espace pour séparer les heures et les minutes.

A noter que, selon que le patient soit orienté dans le circuit court ou le circuit long, le programme demandera la durée soit d'un seul examen, soit de deux. L'utilisateur procède ainsi jusqu'à avoir rentré les informations de tous les patients. Une fois l'opération terminée, la simulation se fait, puis une fenêtre s'ouvre immédiatement.

2 Consultation des résultats de la simulation

La consultation des résultats se fait via une interface graphique. La fenêtre n'est pas redimensionnable. Lors de son ouverture, l'interface a l'apparence suivante :

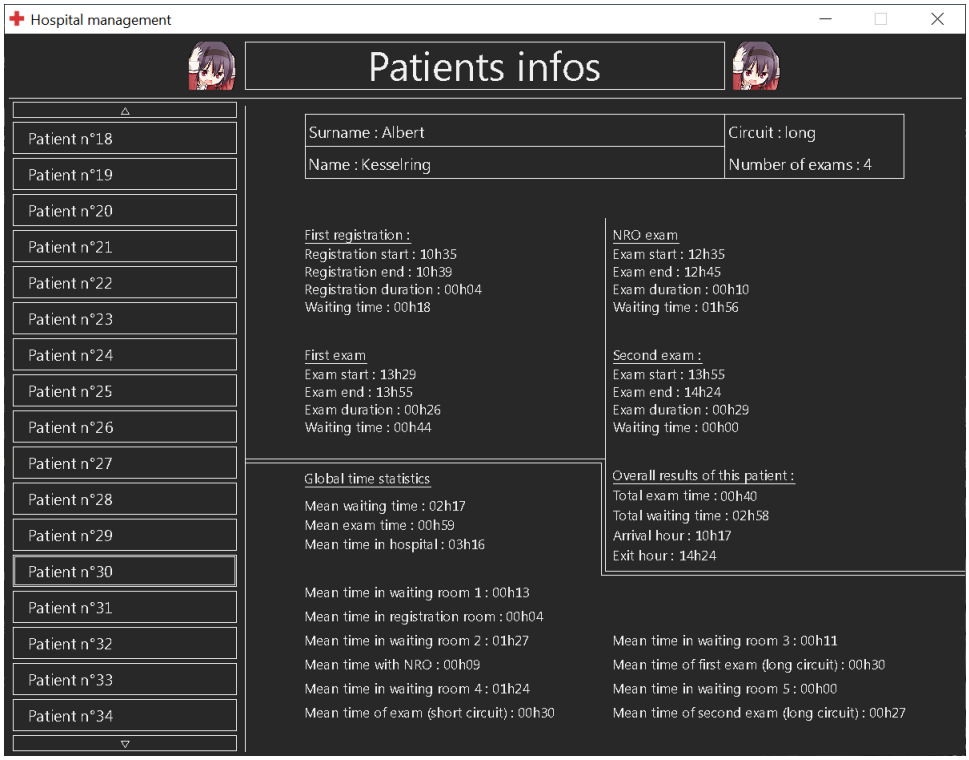

FIGURE 4 – Aperçu de la fenêtre de présentation des résultats

L'utilisateur décide du patient dont il veut afficher les données en cliquant sur les boutons situés sur le bandeau à gauche de la fenêtre.

Il peut ainsi voir la durée, l'heure de début et l'heure de fin de chaque examen. S'il n'a pas passé d'examen complémentaire, le champ reste vierge. Dans le bilan final est inscrit le temps total d'examen, le temps total d'attente et l'heure de sortie du patient. Le cadre supérieur renseigne l'utilisateur sur le nom et le prénom du patient, sur le circuit (court ou long) emprunté par le patient, sur le nombre d'examen passé et sur le temps moyen que le patient a passé à attendre et à se faire examiner au cours de son séjour aux urgences.

En bas de la fenêtre sont inscrites les informations de temps d'attente et de durée d'examen pour tous les patients. Les durées inscrites sont des moyennes. Elles ne dépendent d'aucun patient en particulier et leur affichage est constant.

III. Problèmes rencontrés au cours du projet et perspectives d'amélioration

Les difficultés principales rencontrées au cours de l'avancement du projet ont été de plusieurs ordre ; on peut faire la distinction aux difficultés liées à l'interface graphique et celles liées au back-end.

1 Problèmes liés à la GUI

Pour la génération de la fenêtre de présentation des résultats, les bibliothèques SDL (version 2.0.12) et SDL_ttf (version 2.0.15) ont été utilisées.

Les premières difficultés ont été rencontrées dès l'installation de la bibliothèque, celle-ci n'étant pas comprise de base sur CodeBlocks. Des confusions initiales sur les chemins des fichiers à indiquer au compilateur et sur l'ordre de leur déclaration ont obligé à recommencer plusieurs fois la manœuvre d'installation.

Une fois l'installation réussie, l'autre difficulté a été l'utilisation de la bibliothèque, celle- ci n'ayant pas été étudiée en cours, et son comportement étant relativement différent de ce qui avait été étudié jusque-là en TD.

Le C étant un langage de bas niveau, le moindre affichage d'élément sur la fenêtre requiert plusieurs lignes. Le code servant à créer une fenêtre simple est donc très long. Les mêmes commandes sont sans cesse réutilisées, mais avec des paramètres trop différents pour généraliser l'usage de boucles itératives (même si nous avons essayé d'y recourir le plus possible). Cela rend le code long et fastidieux à écrire.

2 Problèmes liés aux autres aspects du code

De manière générale, à part en ce qui concerne la taille excessive et la non-modularité du code dédié à l'interface graphique, nous n'avons pas été spécialement confronté à des problèmes que nous n'avons pas pu résoudre.

Beaucoup de problèmes sont apparus parce que nous avons commencé à travailler sur le projet extrêmement tôt. Nous avons décidé notre sujet deux mois avant la date butoir, la première version du code (sans interface graphique) a été codée en quelques jours (et quelques nuits blanches), et l'interface graphique en une demi-semaine ; après cela, la correction des bugs a pris une semaine. Le mois et demi restant a été consacré à des consultations occasionnelles du code pour le retravailler et l'optimiser.

Nous avons peut-être eu beaucoup de temps pour les corrections du code, mais finir aussi tôt a également impliqué que nous n'avons pas toujours eu tous les éléments vus en TD à disposition. Par exemple, la première version du code ne contenait pas d'allocation dynamique, et la plupart des problèmes qui sont maintenant résolus grâce à de la gestion dynamique de la

mémoire étaient gérés par la création de grands tableaux (pour le plus grand malheur de nos ordinateurs).

Un autre problème de notre achèvement précoce du prototype du code a entrainé un autre problème, plus indépendant de notre volonté : des détails du projet ont été modifiés après que nous ayons terminé le nôtre. Le désagrément principal a été de nous obliger à retravailler à plusieurs reprises l'architecture de notre interface graphique pour y rajouter des informations nouvellement exigées par le sujet.

Un point difficile à résoudre fut la conception des fonctions de génération aléatoire des durées d'examen et des heures d'arrivée, puisqu'elles faisaient appel à des méthodes statistiques que nous ne connaissions pas. Mais, grâce à l'aide de google et de StackOverflow, ce problème a également été vite résolu.

Quelques autres problèmes ont été causés par notre volonté d'écrire tout le programme, des messages console aux noms de variable, intégralement en anglais pour faire "plus pro" et éviter les limitations du codage ASCII des caractères. N'étant pas anglais et ne parlant pas parfaitement la langue, il n'a bien sûr pas toujours été facile d'écrire les commentaires, mais ce choix a engendré des situations plus gênantes pour nous que réellement nuisibles à la bonne exécution du programme (comme l'usage de noms de variables que nous pensions anglais alors que ce n'était qu'une mauvaise anglicisation d'un mot français).

3 Perspectives d'amélioration

La principale amélioration potentielle qui nous vient à l'esprit serait de créer une base de données à partir du résultat de la simulation. Cette base contiendrait les données des patients et les périodes d'occupation des différentes salles.

En l'état actuel du programme, toute l'entrée de texte se fait depuis la console ; il serait possible d'améliorer l'interface graphique pour que les informations sur les patients soient saisies depuis une fenêtre plutôt que depuis la console.

Une autre amélioration pourrait être un algorithme complémentaire qui envoie un message à la fin de la simulation pour indiquer quels ont été les services les plus congestionnés, et sur quelles plages horaires. Cet algorithme pourrait être améliorée en dressant un tableau des horaires de fréquentation globale de l'hôpital en fonction de l'heure, afin de déterminer où et quand du personnel supplémentaire est nécessaire.

Le programme de base pourrait également être amélioré pour générer le désordre inhérent à tout hôpital : temps de latence lors des échanges de service entre infirmières, pause des médecins de garde, temps de trajet des patients d'une salle à une autre (là où le programme considère le trajet comme instantané)...
Même hors de cette aspect de simulation des problèmes de fonctionnement pourrait être ajouté au programme un concept de priorité de certains patients sur d'autres : si un patient gravement blessé arrive, il passe avant tout le monde.

Cette remarque n'est peut-être pas pertinente, puisque NF05 a pour but de nous apprendre le C et pas un autre langage, mais le sujet de projet consistant en la manipulation fréquente d'objets en utilisant régulièrement les mêmes méthodes, il aurait sans doute été préférable de réaliser le programme en C++ ou dans un autre langage orienté objet.

Conclusion

Ce projet nous a permis de nous améliorer en programmation en mettant en pratique les différents éléments de langage C vus en cours. Nous avons également pu effectuer une première approche du développement d'interface graphique à travers le framework SDL.

Il faut également noter que devoir débugguer un code long de plusieurs centaines de lignes nous a permis de nous entrainer efficacement à la résolution d'erreurs et à la consultation de StackOverflow.

Précisons également que ce projet a été l'occasion de nous initier à LaTeX(même si ça n'a pas de rapport direct avec le langage C).

Un autre intérêt du projet a été une imprécision initiale dans les données de l'énoncé, puis la modification à plusieurs reprises de détails du sujet, nous obligeant à régulièrement nous réadapter.

Annexe

Le code du projet est trouvable en cliquant sur les liens ci-dessous :
- main.c
- input_patient.c (et le header correspondant)
- room_management.c (et le header correspondant)
- sort.c (et le header correspondant)
- gui.c (et le header correspondant)
- struct.h