Université des sciences et de la technologie Houari-Boumédiène
Faculté d'Informatique

« Rapport de Mini projet de TP Algorithmique et Structures de données Programmation C »

Manipulation de structures dynamiques en C Politiques d'allocation de la mémoire à des processus

Réalisé par :

Salmi Sifeddine

Bennaceur Younes

Sous la direction de :

Mme S.Boukhedouma

Mr.A.Lahreche

2023/2024

1-introduction:

Ce rapport présente le projet réalisé dans le cadre du Module ALGO à l'USTHB, mettant l'accent sur la simulation des politiques d'allocation mémoire pour des processus s'exécutant simultanément sur un ordinateur. L'objectif principal de cette simulation est de reproduire dynamiquement le partage et l'organisation de la RAM entre ces processus. En utilisant des structures dynamiques en langage C, notamment des listes chaînées et des files, notre approche vise à explorer les subtilités de la gestion des ressources mémoire lors de l'exécution des programmes. Le projet met en lumière l'importance cruciale d'une gestion efficace des ressources mémoire, et pour atteindre cet objectif, nous simulons diverses stratégies d'allocation telles que First Fit, Best Fit et Worst Fit. Cette simulation constitue une analyse approfondie des mécanismes sous-jacents à la gestion mémoire, offrant une perspective professionnelle sur les dynamiques complexes de l'allocation mémoire en environnement informatique. Nous vous convions à plonger dans cette étude approfondie des politiques d'allocation mémoire, illustrant notre engagement à explorer les subtilités de la gestion des ressources dans un contexte informatique avancé.

Objectifs du TP :

- Acquérir une expertise approfondie dans la manipulation des structures dynamiques en langage C, en mettant l'accent sur les listes chaînées, les files et les piles.
- Mettre en œuvre avec précision et discernement différentes politiques d'allocation de mémoire, telles que "First Fit", "Best Fit" et "Worst Fit", pour gérer simultanément plusieurs programmes.
- Pratiquer de manière rigoureuse l'art de l'écriture de fonctions, en se concentrant sur les aspects liés à la gestion de la mémoire et des processus.
- Concevoir et mettre en œuvre un menu interactif à choix multiples, offrant à l'utilisateur une expérience fluide pour exécuter diverses opérations liées à la gestion de la mémoire.
- S'initier aux subtilités des fonctions d'affichage graphique en langage C, permettant une représentation visuelle claire et conviviale de l'état dynamique de la mémoire.
- Affiner les compétences en communication technique en apprenant à rédiger un compte rendu de travaux pratiques, capturant de manière holistique les étapes du projet et les résultats obtenus

2-Part 1:

Dans cette entreprise de programmation, le projet se concentre sur la simulation de l'organisation et des stratégies d'allocation de mémoire pour des processus parallèles en utilisant le langage de programmation C

Partition de mémoire :

Une partition de mémoire est une plage d'adresses contiguës caractérisée par son point de départ, sa taille, et un état (occupé ou libre). Ce concept permet une gestion dynamique de l'allocation de mémoire en identifiant les zones disponibles et en évitant les conflits potentiels.

Processus :

Un processus est une entité opérationnelle d'un programme, définie par un identifiant exclusif, un instant précis de début, une durée d'exécution spécifique et une taille déterminée. Ces attributs fondamentaux facilitent la gestion algorithmique des processus, favorisant l'ordonnancement, la planification et l'optimisation des ressources du système.

File de processus :

Une file de processus représente une structure de données organisée, servant à stocker les processus en attente d'assignation de mémoire. Elle offre un accès ordonné aux processus en file d'attente, optimisant ainsi l'allocation des ressources système. Cette structure est essentielle dans l'implémentation algorithmique de la gestion des processus et de la mémoire.d

Structure du code:

Le code fourni utilise des structures de données pour représenter les partitions de mémoire, les processus, les files de processus . Il propose également des fonctions facilitant divers aspects de la gestion des ressources mémoire.

Pour les partitions de mémoire, le code met en place des fonctions permettant de créer une liste de partitions. En ce qui concerne les processus, des fonctions sont fournies pour initialiser une file de processus, ajouter un processus à une file, supprimer un processus d'une file, rechercher le premier processus arrivé, supprimer un processus de la memoire, réorganiser les partitions après la suppression d'un processus, et dessiner les partitions à l'aide de la bibliothèque graphique raylib.

Notamment, les fonctions "initfile", "enfiler", "defiler", "TeteFile", "FileVide", et "Affichefile" sont employées pour gérer de manière algorithmique la file de processus. Ces fonctions facilitent

l'initialisation de la file, l'ajout et la suppression d'éléments, l'obtention de la tête de file, la vérification de la vacuité de la file, et l'affichage de son contenu de manière efficace.

L'utilisation de ces structures de données et fonctions offre une approche méthodique et structurée pour simuler les politiques d'allocation mémoire, soulignant l'efficacité et la praticité du code dans la gestion des ressources en environnement informatique.

La fonction createfile :

La fonction "createfile" est implémentée pour générer de manière algorithmique une file de processus en créant un nombre déterminé de processus de façon aléatoire

Les fonctions BestFit, FirstFit et WorstFit :

Les fonctions "BestFit", "FirstFit", et "WorstFit" sont mises en œuvre pour allouer de manière algorithmique de l'espace mémoire aux processus en utilisant respectivement les algorithmes de "Meilleur ajustement", "Premier ajustement" et "Pire ajustement".

La structure partition :

une structure de données pour représenter une partition de mémoire en incluant les attributs adr (pour l'adresse de départ), taille (pour la taille de la partition), temp d'exécution et etat (pour l'état de la partition, où 1 indique qu'elle est occupée et 0 qu'elle est libre).

La structure processus :

représente un processus avec les champs id (identifiant du processus), ia (instant d'arrivée), te (temps d'exécution) et taille (taille du processus).

La structure file :

représente une file de processus avec les pointeurs tete et queue pour le premier et le dernier élément de la file.

Les algorithmes d'allocation de mémoire déterminent comment les partitions de mémoire sont attribuées aux processus. Voici une brève description des trois algorithmes que nous allons explorer :

First Fit :

L'algorithme "First Fit" attribue la première partition libre disponible, dont la taille est adéquate pour accueillir le processus.

Best Fit :

L'algorithme "Best Fit" sélectionne la partition libre ayant la taille minimale nécessaire pour accueillir le processus, avec pour objectif de minimiser le gaspillage d'espace.

Worst Fit :

L'algorithme "Worst Fit" identifie la partition libre ayant la plus grande taille adéquate pour accueillir le processus, visant ainsi à maximiser le gaspillage d'espace.

Structure du code:

on a utiliser les structure suivant :

```
struct memoryPartition
{
    int address;
    int size;
    bool free;
    struct process *allocatedProcess;
    struct memoryPartition *next;
    bool timerState;
    double startTime;
};
struct process
{
    int id;
    int arrivalTime;
    int executionDuration;
    int size;
```

```
    struct process *next;
};

struct Queue
{
    int front, rear;
    int array[MAX_QUEUE_SIZE];
};

struct Stack
{
    struct Queue *data;
    struct Stack *next;
};
```

Le code fourni utilise les structures de données mentionnées précédemment et implémente les trois algorithmes d'allocation de mémoire. Voici une brève description de chaque fonction :

void initializeMemory(struct memoryPartition **head) :

crée une liste chaînée de partitions en générant des adresses et des tailles aléatoires pour chaque partition. Pour chaque partition à créer, un nouveau nœud est créé, avec des valeurs aléatoires pour l'adresse, la taille et l'état (0 pour libre, 1 pour occupée), et est inséré en tête de la liste. La fonction retourne la liste ainsi formée.

void printMemory(struct memoryPartition *memory) :

permet d'afficher les partitions de mémoire contenues dans une liste chaînée. Elle parcourt la liste et affiche les valeurs des champs adresse, taille et état pour chaque nœud.

void initializeProcessesQueue(struct Queue *queue, struct process *processArray) :

initialise une file de processus en la configurant avec une tête et une queue initialisées à NULL.

struct Queue *createQueue() :

crée une file de n processus en générant des valeurs aléatoires pour les champs id, ia, te et taille de chaque processus, puis les ajoute à la file nouvellement créée.

int isEmpty(struct Queue *queue) :

crée une file de n processus en générant des valeurs aléatoires pour les champs id, ia, te et taille de chaque processus, puis les ajoute à la file nouvellement créée.

struct process *dequeue(struct Queue *queue) :

supprime et renvoie le premier processus de la file selon le principe FIFO. Elle vérifie si la file est vide, et si non, elle retire le premier élément, met à jour la tête de la file, et retourne le processus retiré.

void enqueue(struct Queue *queue, struct process *item) ;

ajoute un processus à la file. Elle crée un nouveau nœud avec le processus donné, l'insère à la fin de la file, met à jour la queue de la file en conséquence.

struct process *front(struct Queue *queue) :

renvoie le premier processus de la file sans le supprimer. Elle vérifie si la file est vide, et si non, elle renvoie le processus en tête.

void initializeStack(struct Stack *stack, struct Queue *highPriorityQueue) :

Cette fonction initialise une pile en la mettant à NULL.

int isStackEmpty(struct Stack *stack):

Cette fonction vérifie si la pile est vide. Elle renvoie 1 si la pile est vide et 0 sinon.

void push(struct Stack *stack, struct Queue *queue);

Cette fonction empile un élément sur la pile. Elle prend en paramètres un pointeur vers la pile, une file de processus et une priorité. Elle crée un nouvel élément, lui assigne la priorité et la file de processus, puis le place au sommet de la pile en mettant à jour les pointeurs.

struct Queue *pop(struct Stack *stack):

Cette fonction dépile un élément de la pile. Elle prend en paramètre un pointeur vers la pile. Elle récupère l'élément au sommet de la pile, met à jour les pointeurs pour retirer cet élément de la pile, puis le libère en mémoire. Elle renvoie l'élément défilé.

void firstFit(struct memoryPartition **memory, struct process *currentProcess) :

met en œuvre l'algorithme First Fit, attribuant la première partition libre rencontrée avec une taille adéquate. Elle considère la liste des partitions, la file des processus, le nombre total de processus, une file temporaire, et la table des affectations. L'algorithme sélectionne la première partition libre compatible, ajuste la table des affectations, gère la file des processus, et génère des partitions résiduelles si nécessaire. La fonction retourne la table des affectations actualisée à la fin du processus.

void firstFitUntilFull(struct memoryPartition **memory, struct Queue *processQueue):

La fonction firstFitUntilFull orchestre l'allocation itérative de processus dans les partitions de mémoire en utilisant la stratégie de premier ajustement (First Fit) jusqu'à ce que la mémoire soit pleine ou que la file de processus soit vide. À chaque itération, elle retire un processus de la file à l'aide de dequeue et utilise la fonction firstFit pour son allocation en utilisant la stratégie de premier ajustement. La boucle while continue tant que la mémoire n'est pas pleine et que la file n'est pas vide. Après l'exécution, la fonction affiche un message signalant soit que la mémoire est pleine et aucune allocation supplémentaire n'est possible, soit que tous les processus ont été alloués avec succès si la file est épuisée. En résumé, cette fonction coordonne efficacement l'allocation de mémoire en utilisant la stratégie de premier ajustement dans le contexte de la simulation des politiques d'allocation mémoire.

void bestFit(struct memoryPartition **memory, struct process *currentProcess)

implémente l'algorithme Best Fit pour l'allocation de mémoire, visant à minimiser le gaspillage. Elle prend en compte la liste des partitions disponibles, la file des processus, le nombre total de processus, une file temporaire, et la table des affectations. L'algorithme parcourt les partitions libres, sélectionne celle avec la taille minimale suffisante pour le processus actuel, et ajuste la table des affectations en conséquence. La gestion de la file des processus et la création éventuelle de partitions résiduelles sont également prises en compte, aboutissant à une table des affectations mise à jour

void bestFitUntilFull(struct memoryPartition **memory, struct Queue *processQueue):

La fonction bestFitUntilFull gère l'allocation itérative de processus dans les partitions de mémoire, en utilisant la stratégie Best Fit, jusqu'à ce que la mémoire soit pleine ou que la file de processus soit vide. À chaque itération, elle extrait un processus de la file avec dequeue et utilise la fonction bestFit pour son allocation. La boucle while continue tant que la mémoire n'est pas pleine et que la file n'est pas vide. Après l'exécution, la fonction affiche un message signalant soit que la mémoire est pleine et aucune allocation supplémentaire n'est possible, soit que tous les processus ont été alloués avec succès si la file est épuisée. En résumé, cette fonction coordonne l'allocation de manière efficiente en utilisant la stratégie Best Fit dans le contexte de la simulation des politiques d'allocation mémoire.

void worstFit(struct memoryPartition **memory, struct process *currentProcess)

réalise l'algorithme Worst Fit, une approche d'allocation de mémoire qui vise à maximiser le gaspillage. Elle prend en compte la liste des partitions disponibles (L), la file des processus (F), le nombre total de processus (n), une file temporaire (h), et la table des affectations (T). L'algorithme explore les partitions libres, sélectionne celle avec la plus grande taille pour accommoder le processus en cours, et met à jour la table des affectations en conséquence. La gestion de la file des processus et la création potentielle de partitions résiduelles sont également gérées, aboutissant à une table des affectations mise à jour à la fin de l'allocatio

void worstFitUntilFull(struct memoryPartition **memory, struct Queue *processQueue)

La fonction worstFitUntilFull alloue de manière itérative des processus en utilisant la stratégie d'allocation mémoire Worst Fit jusqu'à ce que la mémoire soit pleine ou que la file de processus soit vide. Elle utilise une boucle while pour extraire des processus de la file et les allouer en mémoire à l'aide de la fonction worstFit. Ce processus se répète tant que la mémoire n'est pas pleine et que la file de processus n'est pas vide. À la fin de la fonction, des messages informatifs sont affichés en fonction du résultat, indiquant si la mémoire est pleine ou si tous les processus ont été alloués avec succès.

Fonction de Dessin:

void drawMemoryPartition(struct memoryPartition *partition, int yPos, bool *timerState);

pour dessiner une seul partition de memoire, avec les donnes de cette partition

void drawMemoryLayout(struct memoryPartition *memory, bool *timerState);
pour dessiner tous les cases de la memoire

void drawMemoryTable(struct memoryPartition *memory);

pour dessiner le tableaux descriptif de l'tetat de memoire.

void drawVerticalQueue(struct Queue *queue, struct process *processArray);

pour dessiner la queue.

void drawVerticalStack(struct Stack *stack);

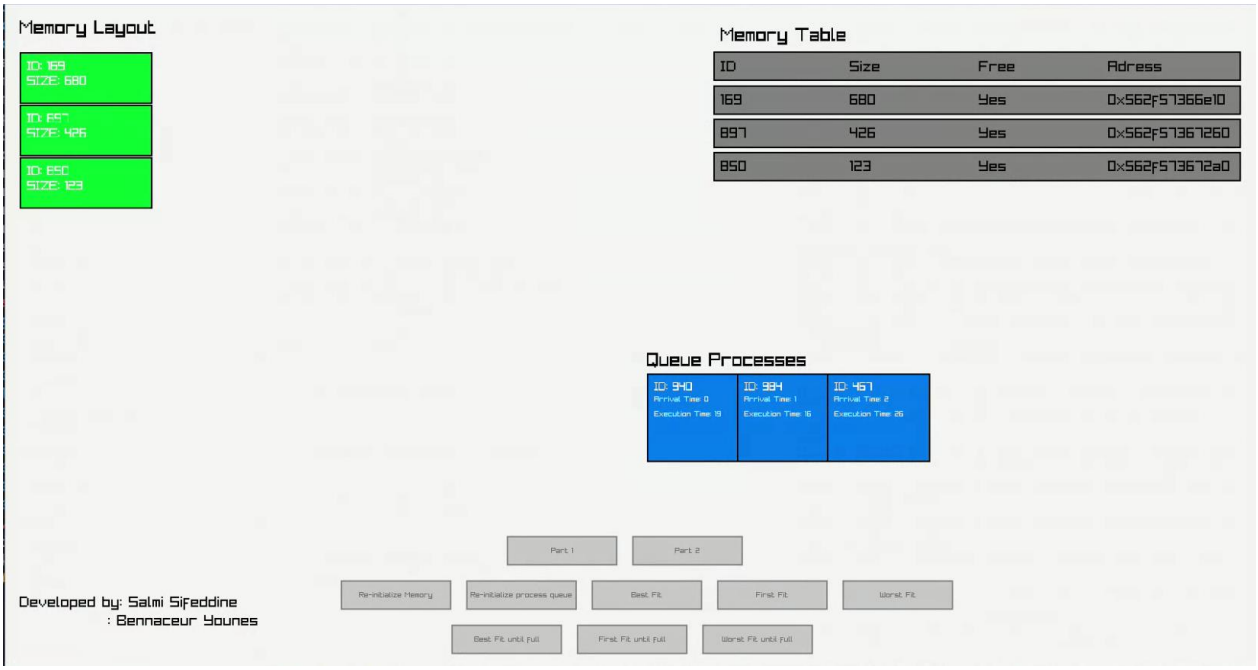
pour dessiner la pile des queues.

prototype

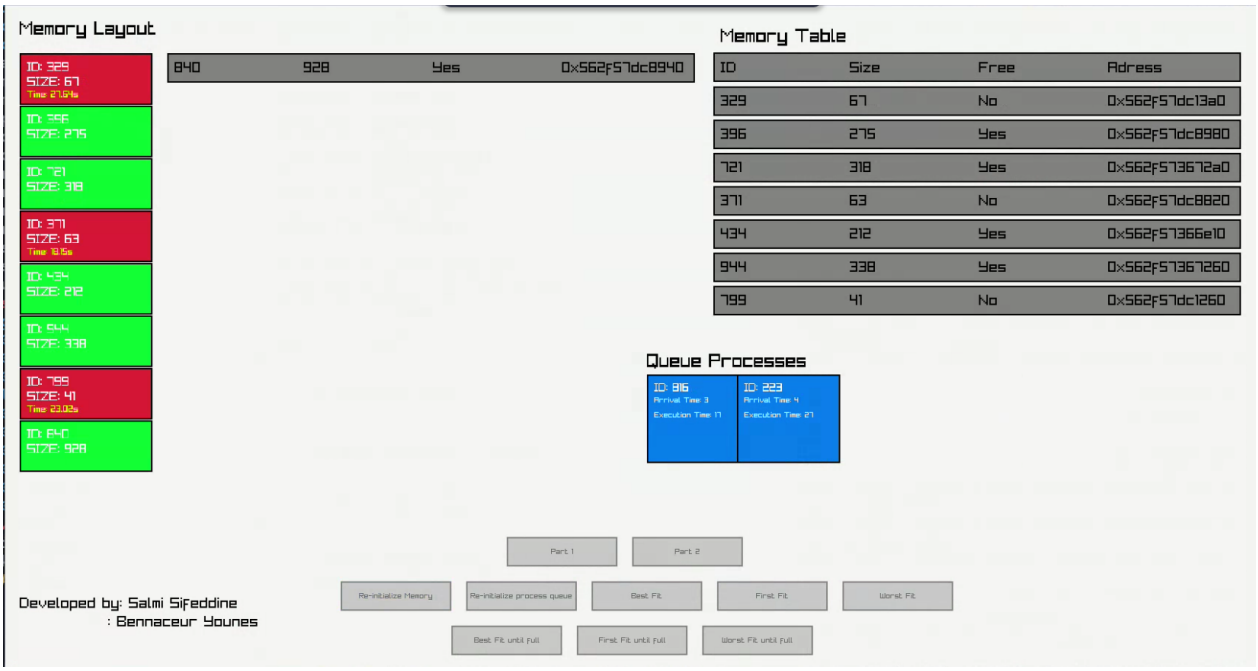
Les principaux composants comprennent la gestion dynamique des partitions de mémoire, des processus, et la mise en œuvre de trois politiques d'allocation de mémoire : First Fit, Best Fit et Worst Fit. Le modèle de mémoire envisage la RAM comme une liste chaînée de partitions, chacune caractérisée par une adresse de départ, une taille en octets et un état indiquant si elle est libre ou occupée. Les processus sont mis en file d'attente en fonction de leur heure d'arrivée, selon une structure First-In-First-Out (FIFO), attendant d'être chargés dans les partitions de mémoire disponibles. Les politiques d'allocation dictent comment les processus sont assignés aux partitions, en tenant compte de facteurs tels que la mémoire résiduelle minimale ou maximale. L'utilisateur interagit avec la simulation grâce à une interface basée sur un menu qui permet des actions telles que l'initialisation des états de la mémoire, l'affichage de représentations textuelles et graphiques de la mémoire, la création de files de processus, le choix des politiques d'allocation, le chargement des processus et la réorganisation de la mémoire après l'achèvement des processus. La représentation graphique utilise la bibliothèque Raylib, améliorant la compréhension des processus dynamiques de gestion de la mémoire par l'utilisateur.

Exemple:

- Initialization et re-inititalizaion du memoire:



- Les method D’allocation temp reel

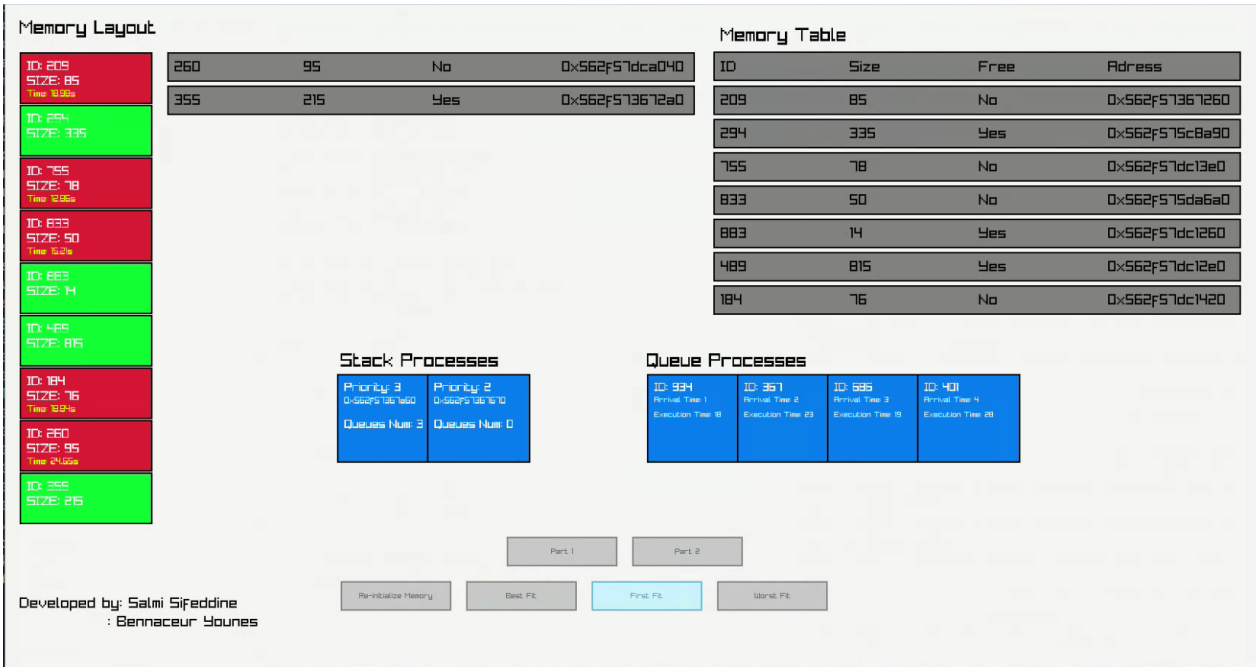


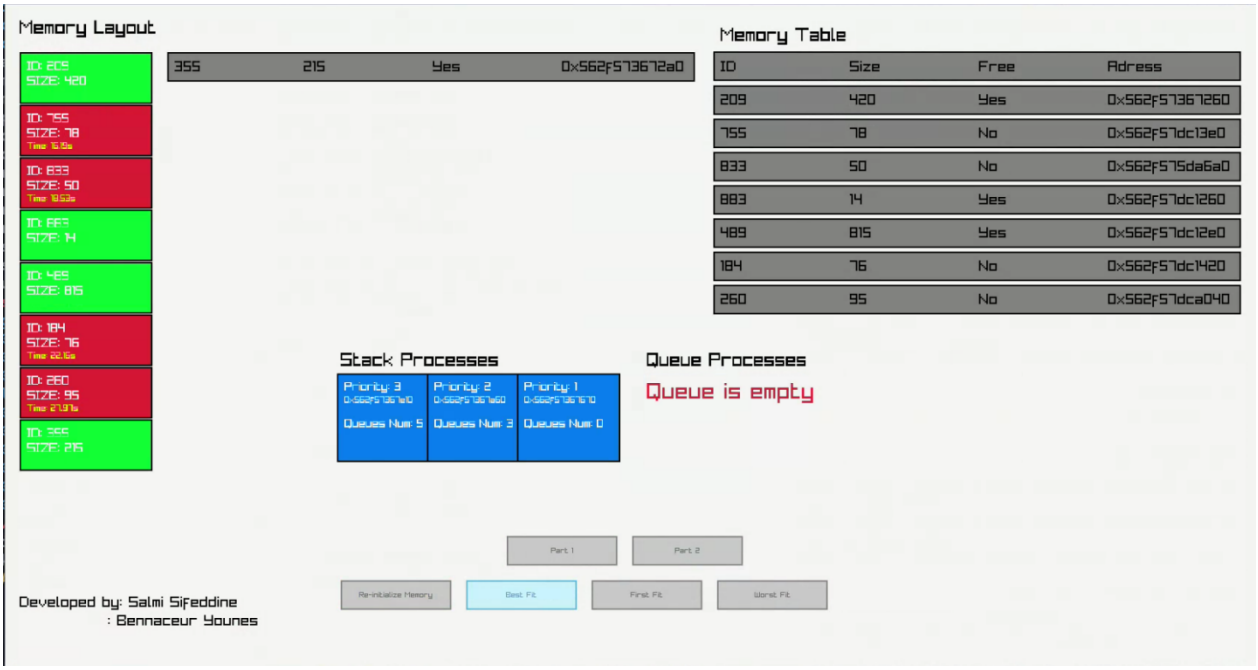
PART II:

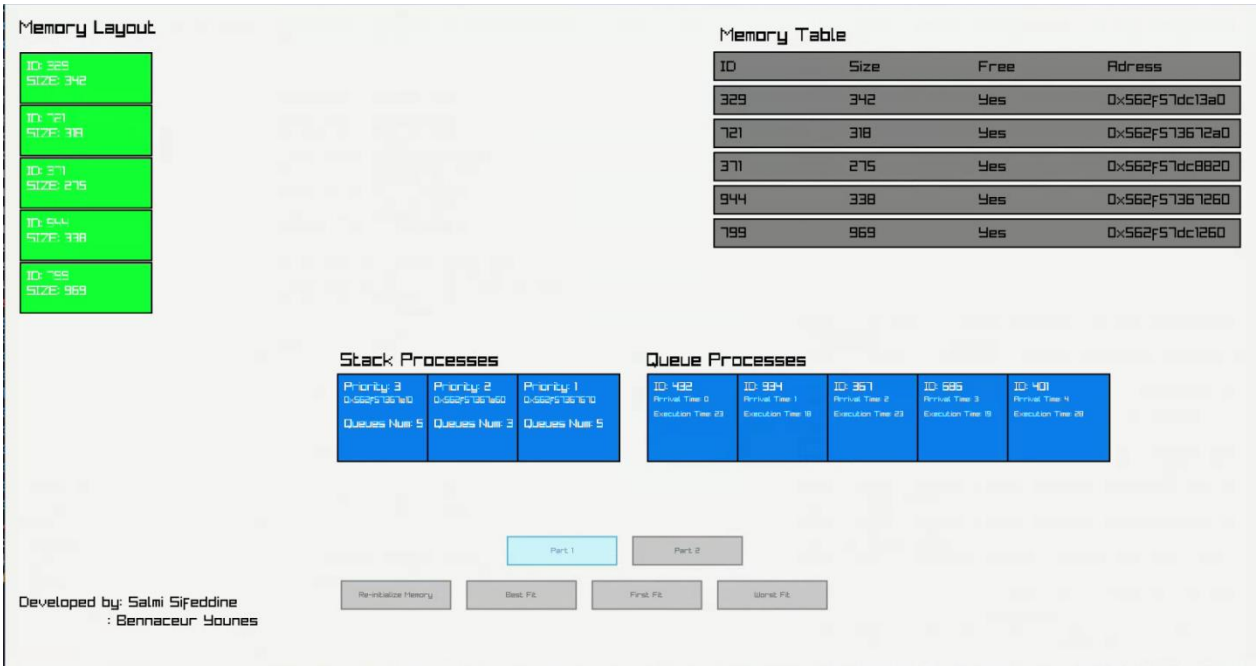
. La deuxième partie du projet introduit des priorités aux processus, créant plusieurs files en fonction des niveaux de priorité et imposant des contraintes sur le chargement des processus en fonction de ces priorités. Dans l'ensemble, le projet constitue une expérience d'apprentissage

complète en programmation C, en structures de données dynamiques et en représentation graphique à l'aide de Raylib

- Les methods d'allocation avec des priorites







Conclusion:

Tout est exécuté dans la fonction principale, où nous avons la déclaration des constantes et la logique de basculement entre différents modes, ainsi que la sélection de différentes méthodes et utilitaires.