Université de Rennes

CYBERSCHOOL
ÉCOLE UNIVERSITAIRE
DE RECHERCHE EN CYBERSÉCURITÉ

Système d'Exploitation Avancée

Rapport du TP

Master 1 Cybersecurité / ISTIC

Ecrit par

Bassirou BADIANE
Ali BA FAQAS

2023/2024

Superviseur

PERREAUDEAU Laurent

Table des matières

Introduction

Ce rapport présente le travail réalisé dans le cadre du Travaux Pratiques sur la mise en œuvre d'un système de gestion de mémoire virtuelle. L'objectif principal de ce TP était de comprendre et d'implémenter les éléments clés d'un système de gestion de mémoire virtuelle, notamment la routine de traitement des défauts de page, un algorithme de remplacement de page, et des fichiers mappés.
Dans un premier temps, nous avons travaillé sur la mise en place d'espaces d'adressage séparés pour permettre l'exécution de plusieurs processus ayant chacun leur table des pages privée. Cette étape a nécessité des modifications dans les méthodes *SaveProcessorState* et *RestoreProcessorState* de la classe *Thread*.
Ensuite, nous avons modifié le chargement des programmes pour qu'il se fasse à la demande, déclenchant un défaut de page lors du premier accès à une page non allouée. Cette modification a impliqué des changements dans le constructeur de la classe *AddrSpace* et la méthode *StackAllocate*.
Enfin, nous avons mis en œuvre une routine de traitement des défauts de page et un algorithme de remplacement de page. Ces deux éléments sont essentiels pour gérer les situations où la mémoire physique est pleine et un processus veut charger en mémoire une page absente de la mémoire physique.
Ce rapport détaille les différentes étapes de notre travail, les défis rencontrés et les solutions apportées pour répondre aux exigences du TP

1 Mise au point du projet

Tout d'abord, nous nous sommes arrêtés sur la partie Gestion de mémoire virtuelle et par conséquent on a pas pu touché à la dernière partie qui parle des fichiers mappées. Pour la première partie notre programme fonctionne normalement lorsque le nombre de pages physiques est assez grand pour ne pas requisitionner une autre page physique lors de la traduction d'adresses. Cependant si on réduit notoirement le nombre de pages physiques, à 8 par exemple, on rencontre quelques difficultés avec la méthode *EvictPage* bien que le programme *hello* continue à fonctionner avec cette méthode. On croit avoir identifié le problème mais nous avons malheureusement pas pu le résoudre malgrè le fait qu'on a passer pas mal de temps là dessous. Notre problème proviendrait d'une erreur sur la gestion de la mémoire swap dans la fonction *EvictPage*.

2 Explication des fonctions principales de traitement de page

La routine de traitement des défauts de page est chargée d'allouer une page physique, de la remplir, puis de redonner la main au thread qui a provoqué le défaut de page. Dans cette section, nous allons expliquer en quelques phrases les algorithmes d'implémentations des fonctions suivantes. *PageFault*, *AddPhysicalToVirtualMapping* et *EvictPage*.

2.1 Explication de l'implémentation de la fonction *EvictPage*

La fonction EvictPage est responsable de l'algorithme de remplacement de page. Elle est appelée lorsqu'il n'y a pas de pages physiques libres disponibles pour être utilisées. L'algorithme utilisé est l'algorithme de l'horloge (clock algorithm). La fonction commence par une boucle infinie, qui parcourt les pages physiques dans un ordre circulaire en utilisant la variable i_clock. À chaque itération, la fonction incrémente la valeur de i_clock modulo le nombre total de pages physiques pour garantir la circularité.
À chaque itération, la fonction vérifie si la page physique actuelle n'est pas verrouillée en vérifiant la valeur du champ locked de la structure tpr. Si la page n'est pas verrouillée, la fonction vérifie ensuite si le bit de référence de la table de traduction de la page virtuelle associée à la page physique est défini. Cela est vérifié en utilisant la fonction getBitU de la table de traduction.
Si le bit de référence est défini, la fonction réinitialise le bit en utilisant la fonction clearBitValid de la table de traduction. Sinon, la page physique est sélectionnée pour être évictée.
Avant de libérer la page physique, la fonction vérifie si le bit de modification de la table de traduction est défini en utilisant la fonction getBitM. Si le bit de modification est défini, cela signifie que la page a été modifiée, donc la fonction sauvegarde la page dans le fichier de swap en appelant la fonction appropriée et définit le bit d'échange en utilisant la fonction setBitSwap de la table de traduction.

Enfin, la fonction retourne le numéro de la page physique évictée.

2.2 Explication de l'implémentation de la fonction *AddPhysicalToVirtualMapping*

La fonction AddPhysicalToVirtualMapping est utilisée pour établir une correspondance entre une page physique libre et une page virtuelle spécifique. Elle est appelée lorsque la mémoire virtuelle nécessite une nouvelle page physique pour stocker des données. La fonction commence par rechercher une page physique libre en appelant la fonction FindFreePage. Si une page libre est trouvée, la fonction récupère son numéro et la marque comme verrouillée en définissant le champ locked de la structure tpr correspondante à true.
Ensuite, la fonction met à jour la table de traduction de la page virtuelle associée à la page physique en utilisant la fonction setPhysicalPage de la table de traduction de l'espace d'adressage du propriétaire. La fonction met également à jour la structure tpr en marquant la page physique comme utilisée en définissant le champ free à false, en enregistrant le numéro de la page virtuelle dans le champ virtualPage et en enregistrant le propriétaire (l'espace d'adressage) dans le champ owner.
Enfin, la fonction retourne le numéro de la page physique allouée.

2.3 Explication de l'implémentation la fonction *PageFault*

La fonction pageFault est responsable de la gestion des fautes de page. Lorsqu'un accès à une page virtuelle provoque une faute de page, cette fonction est appelée pour résoudre la faute. La fonction commence par vérifier si la page virtuelle associée à la faute de page a déjà une correspondance avec une page physique en utilisant la fonction isPageMapped de l'espace d'adressage du propriétaire. Si la page est déjà mappée, cela signifie qu'il s'agit d'une faute de page mineure et la fonction termine sans rien faire.
Si la page virtuelle n'est pas mappée, la fonction appelle la fonction AddPhysicalToVirtualMapping pour allouer une nouvelle page physique et établir une correspondance avec la page virtuelle. Si l'allocation échoue, la fonction retourne un code d'erreur.
La fonction gère également les fautes de page majeures. Si le bit de swap de la page virtuelle est défini, cela signifie que la page a été évincée précédemment. La fonction appelle alors la fonction loadPageFromSwap pour charger la page depuis le fichier de swap.
Si le bit de swap n'est pas défini, la fonction utilise l'algorithme de remplacement de page en appelant la fonction EvictPage pour sélectionner une page à évicter. Ensuite, elle appelle la fonction loadPageFromSwap pour charger la page depuis le fichier de swap.

3 Illustration des résultats

Nous avons testé notre projet avec plusieurs programmes, et les tests étaient les suivants :

3.1 Routine de traitement de défaut de pages sans réquisition avec un ou plusieurs threads

Dans cette section, le nombre de pages physiques dans le fichier de configuration est fixé à 400 pages, il y a donc toujours des pages libres en mémoire réelle donc pas d'appel à la function Evictpage.
Le programme semble fonctionner parfaitement. Voici quelques résultats de l'exécution des programmes shell.c, sort.c et hello.c qui lancent un seul thread, mais aussi rdv.c, productor.c qui lancent plusieurs threads.

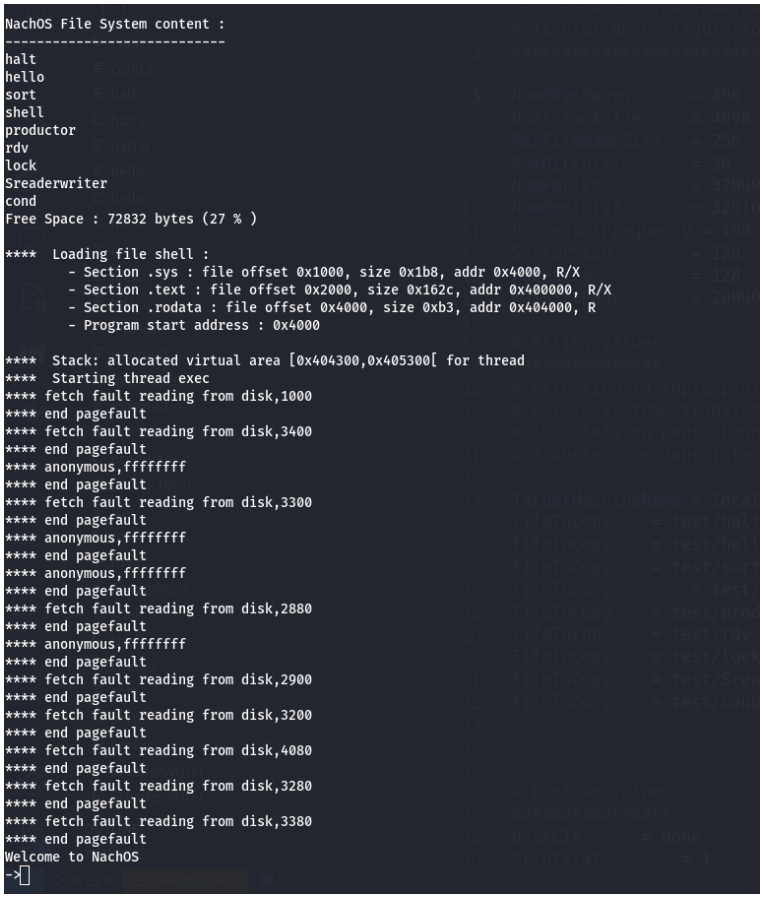

FIGURE 1 – Shell program.

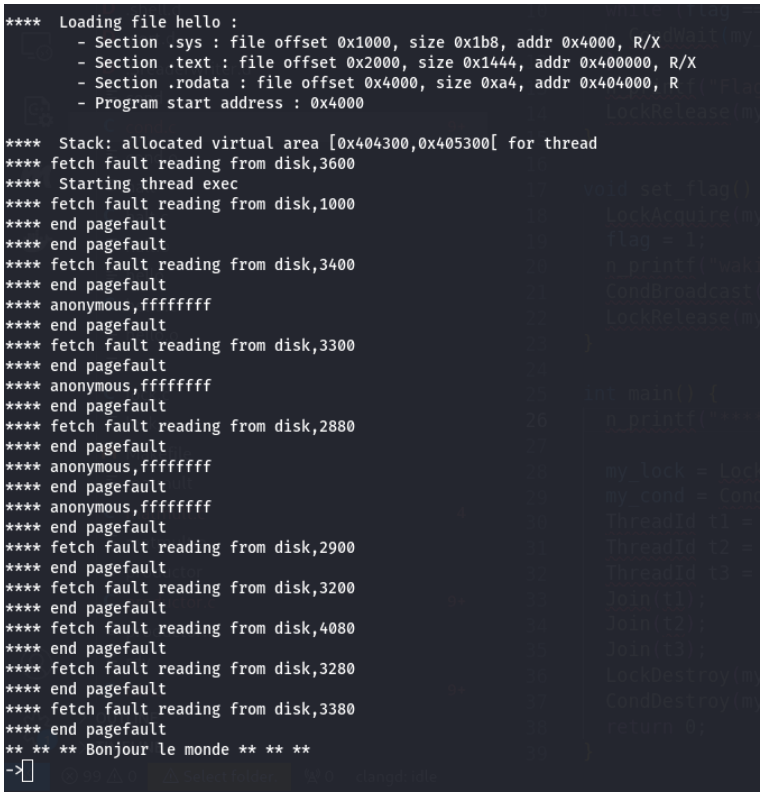

Figure 2 – Hello.c.

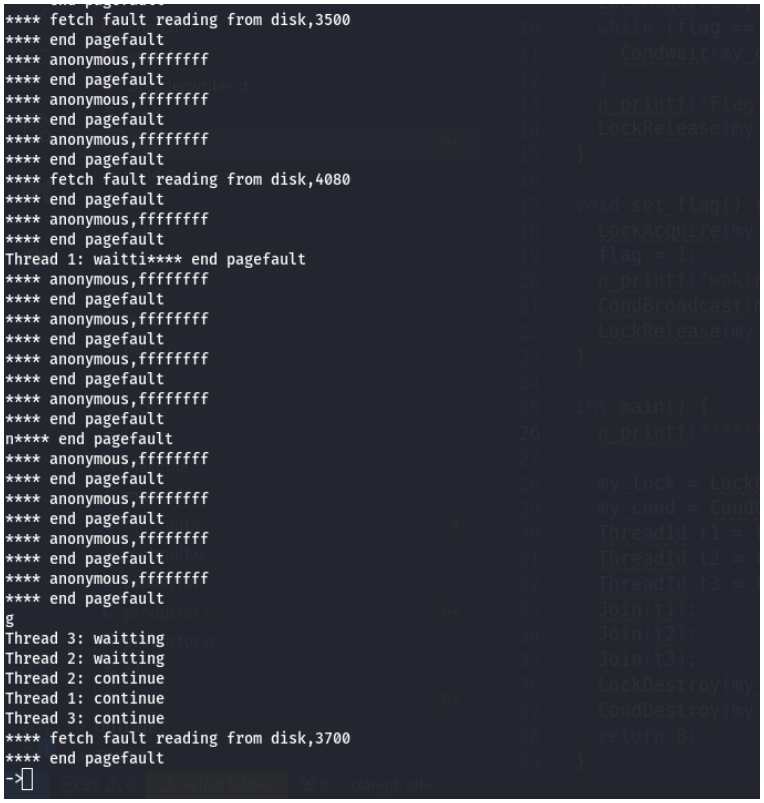

Figure 3 – rdv program.

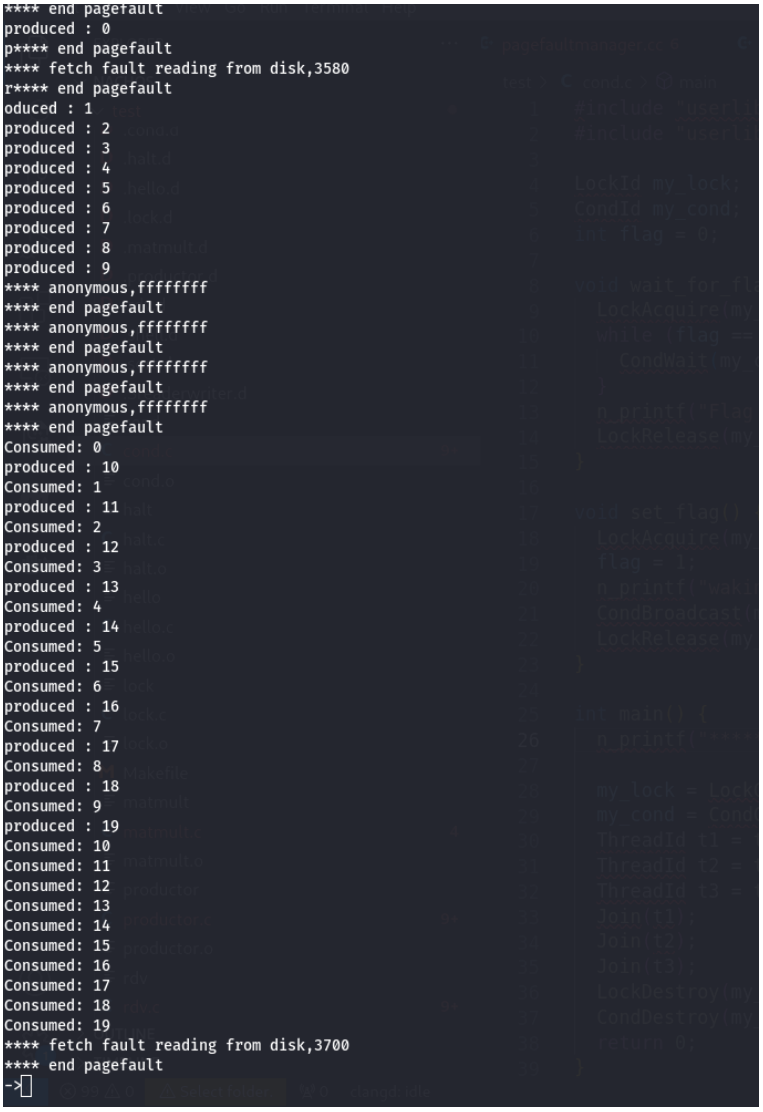

Figure 4 – productor and consumer program.

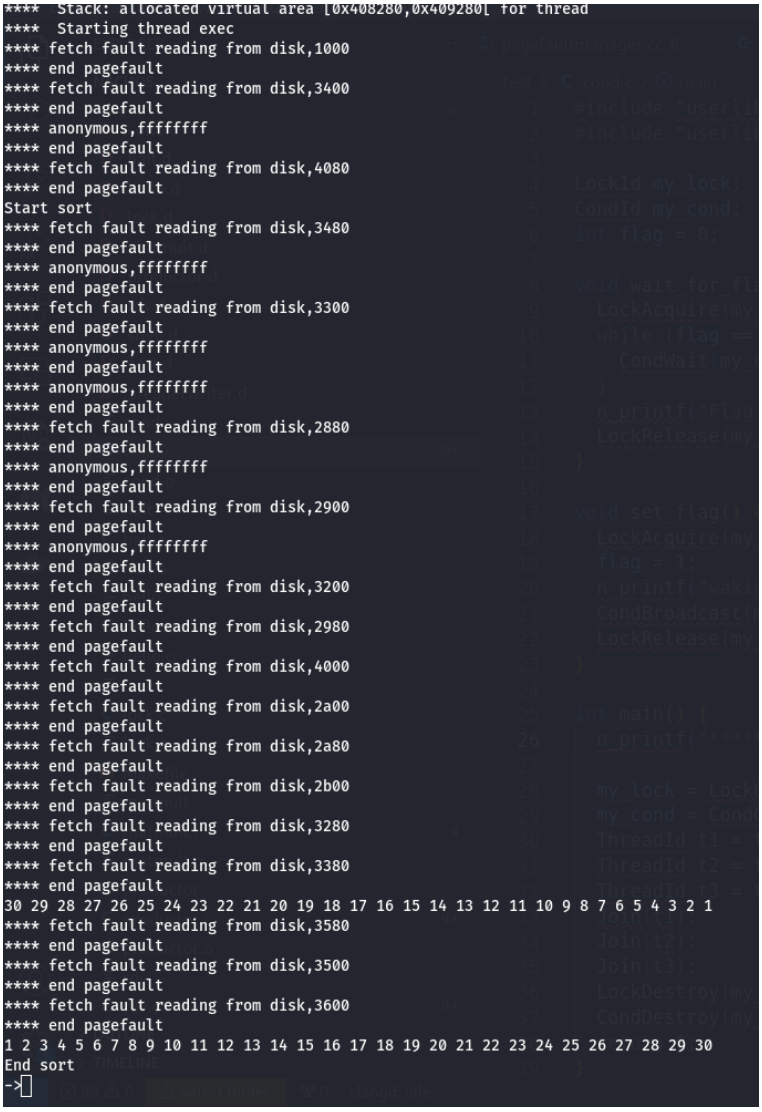

FIGURE 5 – sort program.

3.2 Routine de traitement avec réquisition

Dans cette section, nous avons réduit le nombre de pages physiques dans le fichier de configuration à 10, de sorte qu'il n'y ait pas suffisamment de pages en mémoire réelle, nécessitant ainsi l'appel de la méthode *EvictPage*.
Au cours des tests, les fichiers hello.c et shell.c ont fonctionné correctement. Comme vous pouvez le voir dans la figure suivante, nous avons réussi à évincer une nouvelle page et le programme fonctionne et se termine correctement.

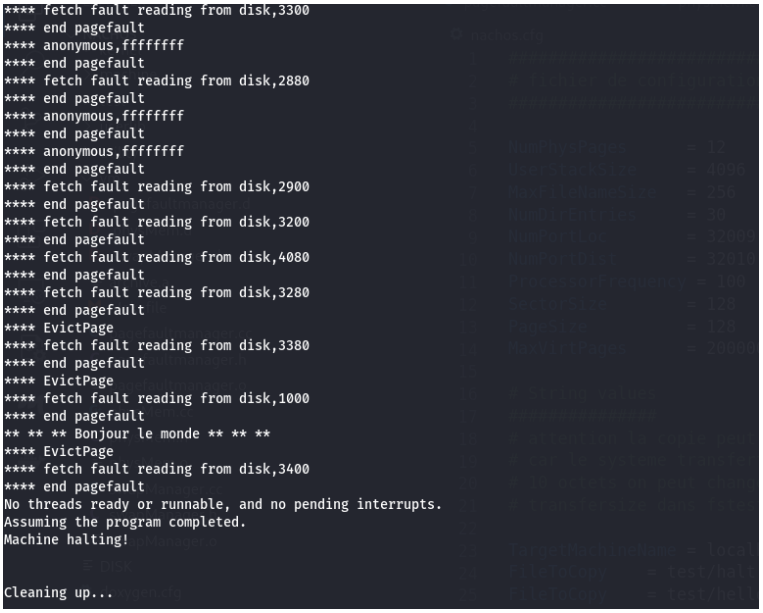

FIGURE 6 – sort program.

Cependant, lorsque nous avons essayé d'exécuter rdv.c ou sort.c, nous avons rencontré l'erreur illustrée dans la figure suivante. Nous pensons que nous avons peut-être commis une erreur dans la fonction EvictPage, mais nous croyons également que ce ne sera pas quelque chose d'énorme car nous pensons que notre fonction EvictPage respecte l'algorithme que nous avons programmé en TD.

FIGURE 7 – sort program.

Conclusion

Ce projet sur la mise en œuvre d'un système de gestion de mémoire virtuelle a été une expérience extrêmement instructive pour nous. Il nous a permis de plonger en profondeur dans les mécanismes internes de la gestion de la mémoire et de comprendre les défis associés à la mise en œuvre d'un tel système.

Les défis rencontrés tout au long du projet n'ont pas été simples à résoudre. Chaque étape, qu'il s'agisse de la mise en place d'espaces d'adressage séparés, du chargement des programmes à la demande, ou de l'implémentation de la routine de traitement des défauts de page et de l'algorithme de remplacement de page, a présenté ses propres obstacles. Cependant, ces défis nous ont offert l'opportunité d'apprendre et de comprende partiellement comment la mémoire est gérée par le système d'exploitation.

Malgré les difficultés, nous sommes parvenus à surmonter les obstacles majeurs sauf un qui nous a empêché de finalement totalement la partie sur la gestion de mémoire virtuelle. Néanmoins, nous sommes satisfait de ce que nous avons réussi à accomplir sur ce projet. En somme, ce projet a été une aventure enrichissante et éducative.