# Mécanisme d’appel et de processus

Lorsque qu’un programme fait un *system call,* le system appel la fonction *ExceptionHandler* du fichier *exception.cc*.

# Gestion de threads et de processus

## 1.1 Indiquer ce qui doit être sauvegarder lors d’un changement de context

Essentiellement les registres du processeur

## 1.2 Quelle variable est utilisée pour mémoriser la liste des threads prêts à s’exécuter ? Est-ce que le thread élu (actif) appartient à cette liste ? Comment accéder à ce thread ?

On utilise la Listint *readyList* pour stocker les threads. Le thread actif n’appartient plus à cette liste car pour l’obtenir il faut passer par la méthode *findNextToRun qui* effectue un *remove* sur *readyList* pour renvoyer le nouveau thread.

## 1.3 A quoi sert la variable *g_alive* ? Quelle est la différence avec le champ *readyList* de l’objet g_scheduler ?

La variable *g_alive* stocke l’ensemble de tous les threads du system contrairement à *readyList* qui contient seulement ceux qui souhaitent être run. Le *g_scheduler* est quant à lui un objet qui ordonnance les threads et permet aussi d’effectuer un changement de contexte avec la méthode *SwitchTo*.

## 1.4 Comment se comportent les routines de gestion de listes vis à vis de l’allocation de mémoire ? Est-ce qu’elles se chargent d’allouer/désallouer les objets chaînés dans la liste ? Pourquoi ?

Non les listes ne se préoccupes pas de l’allocation mémoire de ses éléments car elle ne garde en mémoire que le pointeur vers l’objet suivant.

## 1.5 A quel endroit est placé un objet thread quand il est bloqué sur un sémaphore ?

Les sémaphores contiennent un compteur et une file d’attente donc quand un thread est bloqué il est placé dans cette file d’attente.

## 1.6 Comment faire en sorte qu’on ne soit pas interrompu lors de la manipulation des structures de données du noyau ?

On interdit les interuptions, pour cela on utilise la méthode *SetStatut* de la classe *Interrupt* à laquelle on accède par l’objet *g_machine*.

### 1.7 A quoi sert la méthode SwitchTo de l'objet g scheduler ? Quel est le rôle des variables thread context et simulator context de l'objet thread ? Que font les méthodes SaveSimulatorState et RestoreSimulatorState ? Que devront (à terme) faire les méthodes SaveProcessorState et RestoreProcessorState de l'objet thread ?

La fonction *SwithTo* effectue un changement de thread courant avec les changements de contexte que cela implique.

Les variables de context servent à mémoriser l'état des registre, soit du thread soit du simulateur MIPS.

Les méthodes serve à sauver et restaurer l'état du simulateur MIPS.

Lorsque le processus est interrompu, au moins une autre entité va utiliser les registres actuellement utilisés par le processus. SaveProcessorState et RestoreProcessorState servent donc à enregistrer puis récupérer les données des registres dans la mémoire pour que le jour où le processus reprendra, il n'e puisse pas s'apercevoir qu'il a été interrompu.

### 1.8 Expliquer l'utilité du champ type de tous les objets manipulés par le noyau (sémaphores, tâches, threads, etc.).

Il sert a vérifier si l'objet récupéré est bien du type attendu (souvent obtenu avec un id donc on ne peut pas connaître le type précis sans cette variable)

## Environnement de développement

### 1.1 Lister les outils offerts par Nachos pour la mise au point des programmes utilisateur. Comment par exemple visualiser toutes les opérations effectuées par la machine MIPS émulée ?

??? *(pas compris de quoi on parle ...)*

On peut visualiser toutes les opérations effectuées par la machine MIPS en compilant le programme avec l'option -g ou en allant regarder le fichier *core* généré.

### 1.2 Peut-on utiliser l'utilitaire gdb pour mettre au point le code de Nachos ? Le lancer et visualiser le contenu de différentes variables du noyau.

On peut seulement s'en servir pour mieux comprendre un plantage de l'exécution du programme. Par contre on peut en effet accéder au contenu d'une variable.

### 1.3 Peut-on utiliser gdb pour mettre au point les programmes utilisateur ? Expliquer.

Non car gdb fait du débogage sur l 'exécution d'un programme.