Projet système
Master 1

R. Blanc A.Castel C.Eymond Laritaz M.Garnier

UFR IM^{2}AG
Université Grenoble Alpes

Jeudi 17 Janvier 2019

Outline

Appels systèmes

- void PutChar(char c);
- void PutString(char *s);
- char GetChar();
- void GetString(char *s, int n);
- void PutInt(int n);
- void GetInt(int *n);

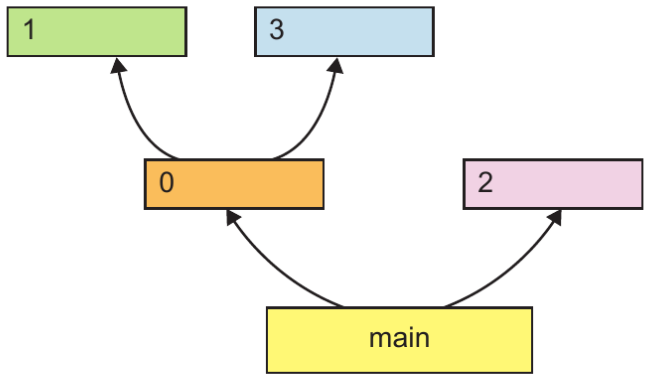
Choix d'implémentation

- Limite de chaîne de caractères traités de 200
- Les '\n' et les 'EOF' sont pris en compte
- La fonction PutString n'est pas synchrone

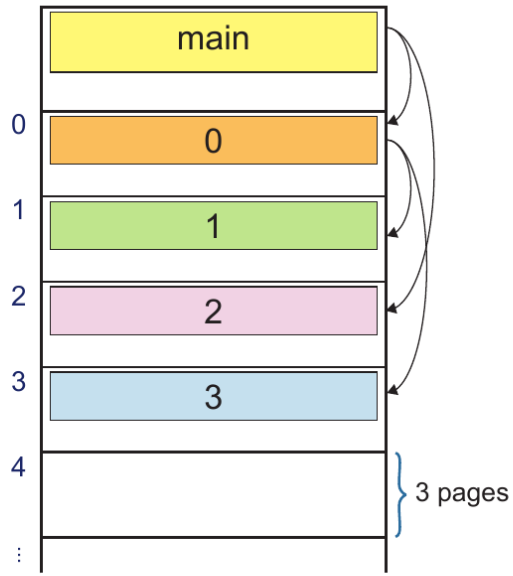
Choix d'implémentation

- Le nombre de Threads est limité à 50
- L'emplacement dans la pile des Threads est géré avec une Bitmap
- 3 pages sont allouées à chaque Thread utilisateur

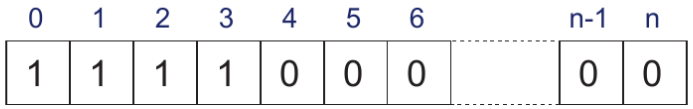
Illustration du placement des Threads



Arborescence
de création des threads

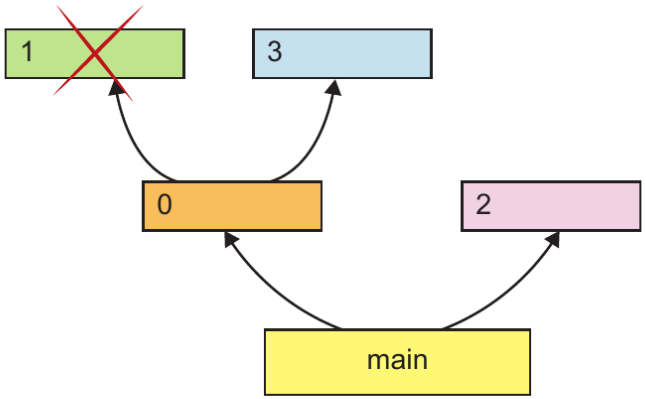


Organisation mémoire
des piles de threads

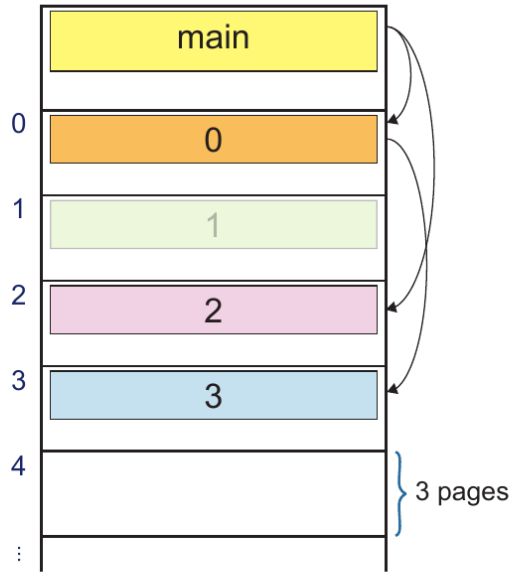


BitMap
blocs mémoire occupés pour les piles de threads

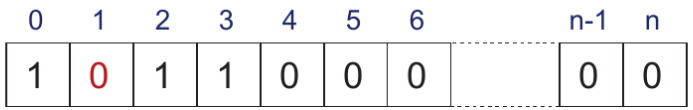
Illustration du placement des Threads



Arborescence
de création des threads

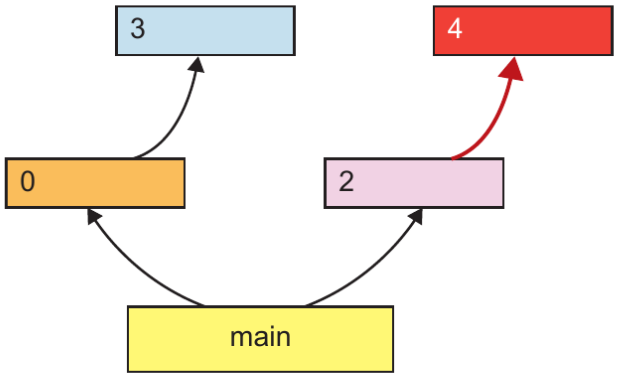


Organisation mémoire
des piles de threads

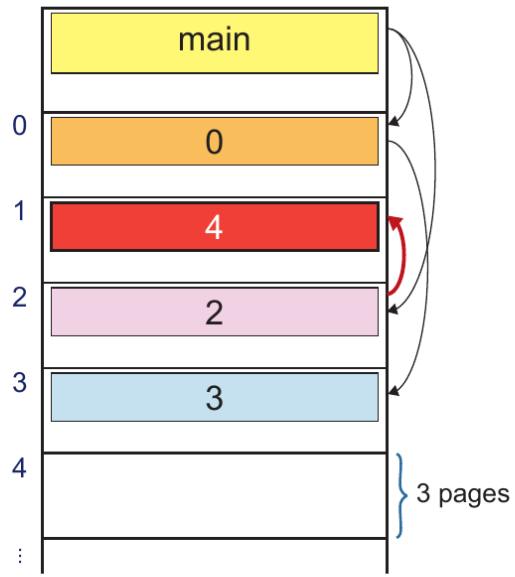


BitMap
blocs mémoire occupés pour les piles de threads

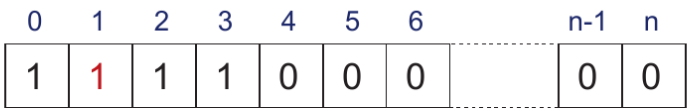
Illustration du placement des Threads



Arborescence
de création des threads



Organisation mémoire
des piles de threads



BitMap
blocs mémoire occupés pour les piles de threads

Appels systèmes

- int UserThreadCreate(void f(void *arg), void *arg);
  - Créé un Thread utilisateur, qui exécute la fonction "f" avec les arguments "arg"
  - Retourne l'identifiant du Thread créé

Appels systèmes

- int UserThreadCreate(void f(void *arg), void *arg);
  - Créé un Thread utilisateur, qui exécute la fonction "f" avec les arguments "arg"
  - Retourne l'identifiant du Thread créé
- void UserThreadExit();
  - Termine un Thread utilisateur

Appels systèmes

- int UserThreadCreate(void f(void *arg), void *arg);
  - Créé un Thread utilisateur, qui exécute la fonction "f" avec les arguments "arg"
  - Retourne l'identifiant du Thread créé
- void UserThreadExit();
  - Termine un Thread utilisateur
- void UserThreadJoin(int tid);
  - Attend la terminaison du Thread utilisateur "tid"

Choix d'implémentation

- L'identifiant d'un Thread est unique, il n'est jamais réutilisé

Choix d'implémentation

- L'identifiant d'un Thread est unique, il n'est jamais réutilisé
- La gestion de la terminaison des Threads est laissée à l'utilisateur, sauf pour le main

Choix d'implémentation

- L'identifiant d'un Thread est unique, il n'est jamais réutilisé
- La gestion de la terminaison des Threads est laissée à l'utilisateur, sauf pour le main
- Un Thread peut faire un `UserThreadJoin` sur n'importe quel Thread créé lors de l'exécution du programme

Choix d'implémentation

- L'identifiant d'un Thread est unique, il n'est jamais réutilisé
- La gestion de la terminaison des Threads est laissée à l'utilisateur, sauf pour le main
- Un Thread peut faire un `UserThreadJoin` sur n'importe quel Thread créé lors de l'exécution du programme
- C'est à l'utilisateur d'utiliser correctement `UserThreadJoin`

Choix d'implémentation

- L'identifiant d'un Thread est unique, il n'est jamais réutilisé
- La gestion de la terminaison des Threads est laissée à l'utilisateur, sauf pour le main
- Un Thread peut faire un `UserThreadJoin` sur n'importe quel Thread créé lors de l'exécution du programme
- C'est à l'utilisateur d'utiliser correctement `UserThreadJoin`
  - Par exemple, l'utiliser sur un identifiant de Thread incorrect donnera lieu à un comportement indéfini.

Autres ajouts

Synchronisation

- Nouveau type `semaphore_t` au niveau utilisateur, avec les appels systèmes qui permettent de le manipuler.

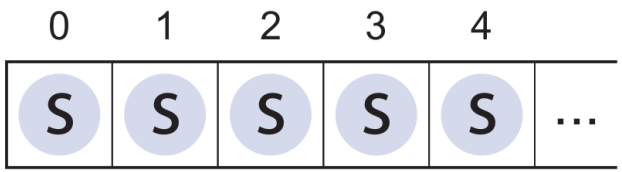


tableau de sémaphores

Autres ajouts

Synchronisation

- Nouveau type `semaphore_t` au niveau utilisateur, avec les appels systèmes qui permettent de le manipuler.
  - semaphore_t Sem_Init(int nbJetons);

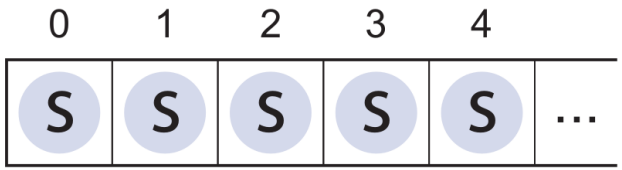


tableau de sémaphores

Autres ajouts

Synchronisation

- Nouveau type `semaphore_t` au niveau utilisateur, avec les appels systèmes qui permettent de le manipuler.
  - semaphore_t Sem_Init(int nbJetons);
  - int Sem_P(semaphore_t s);

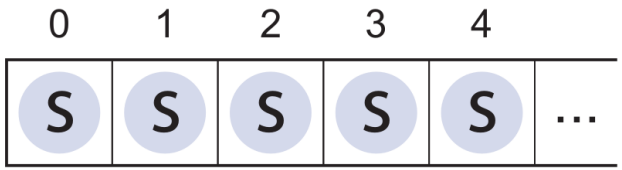


tableau de sémaphores

Autres ajouts

Synchronisation

- Nouveau type `semaphore_t` au niveau utilisateur, avec les appels systèmes qui permettent de le manipuler.
  - semaphore_t Sem_Init(int nbJetons);
  - int Sem_P(semaphore_t s);
  - int Sem_V(semaphore_t s);

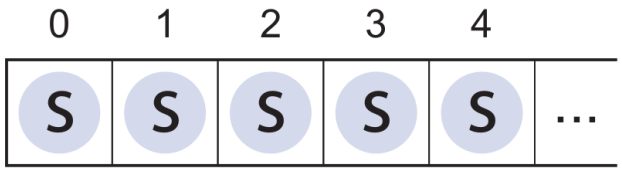


tableau de sémaphores

Autres ajouts

Synchronisation

- Nouveau type `semaphore_t` au niveau utilisateur, avec les appels systèmes qui permettent de le manipuler.
  - semaphore_t Sem_Init(int nbJetons);
  - int Sem_P(semaphore_t s);
  - int Sem_V(semaphore_t s);
  - int Sem_GetValue(semaphore_t s);

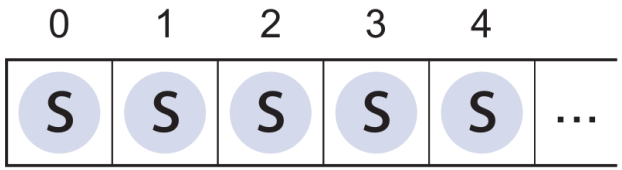


tableau de sémaphores

Autres ajouts

Synchronisation

- Nouveau type `semaphore_t` au niveau utilisateur, avec les appels systèmes qui permettent de le manipuler.
  - semaphore_t Sem_Init(int nbJetons);
  - int Sem_P(semaphore_t s);
  - int Sem_V(semaphore_t s);
  - int Sem_GetValue(semaphore_t s);
  - int Sem_Destroy(semaphore_t s);

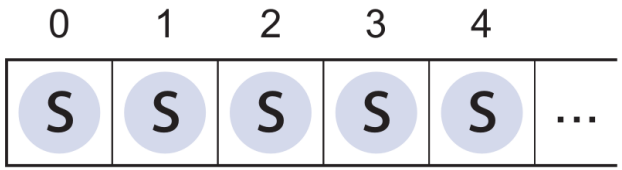


tableau de sémaphores

Autres ajouts

Synchronisation

- Nouveau type `semaphore_t` au niveau utilisateur, avec les appels systèmes qui permettent de le manipuler.
  - semaphore_t Sem_Init(int nbJetons);
  - int Sem_P(semaphore_t s);
  - int Sem_V(semaphore_t s);
  - int Sem_GetValue(semaphore_t s);
  - int Sem_Destroy(semaphore_t s);
- Fonctionnement avec une table de sémaphores noyau

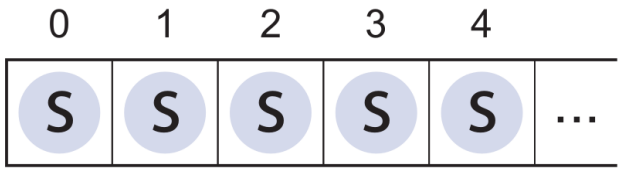


tableau de sémaphores

Pagination

Mémoire virtuelle

- Frame Provider

Pagination

Mémoire virtuelle

- Frame Provider
- Politiques d'allocation

Pagination

Mémoire virtuelle

- Frame Provider
- Politiques d'allocation
  - FIRST_FREE_FRAME

Pagination

Mémoire virtuelle

- Frame Provider
- Politiques d'allocation
  - FIRST_FREE_FRAME
  - RANDOM_FREE_FRAME

Multi-Processus

Appel système

- Création d'un nouveau type utilisateur pid_t

Multi-Processus

Appel système

- Création d'un nouveau type utilisateur pid_t
- pid_t ForkExec(char* filename)
  - Crée un nouveau processus qui exécute le programme contenu dans le fichier filename.
  - Renvoie le pid_t correspondant au nouveau processus

Multi-Processus

Appel système

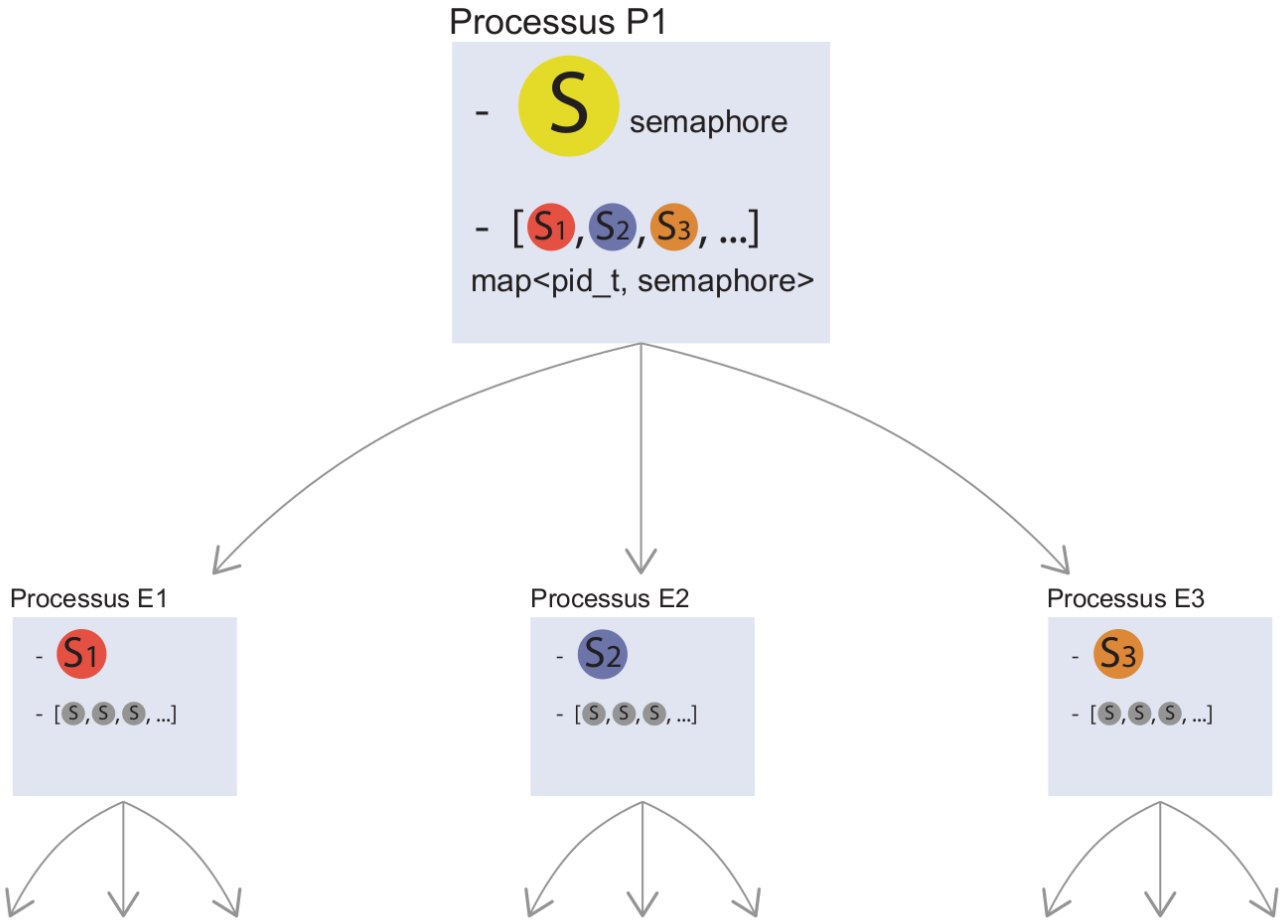
- Création d'un nouveau type utilisateur pid_t
- pid_t ForkExec(char* filename)
  - Crée un nouveau processus qui exécute le programme contenu dans le fichier filename.
  - Renvoie le pid_t correspondant au nouveau processus
- void waitpid(pid_t p)
  - Attend la terminaison du processus p

Multi-Processus

Appel système

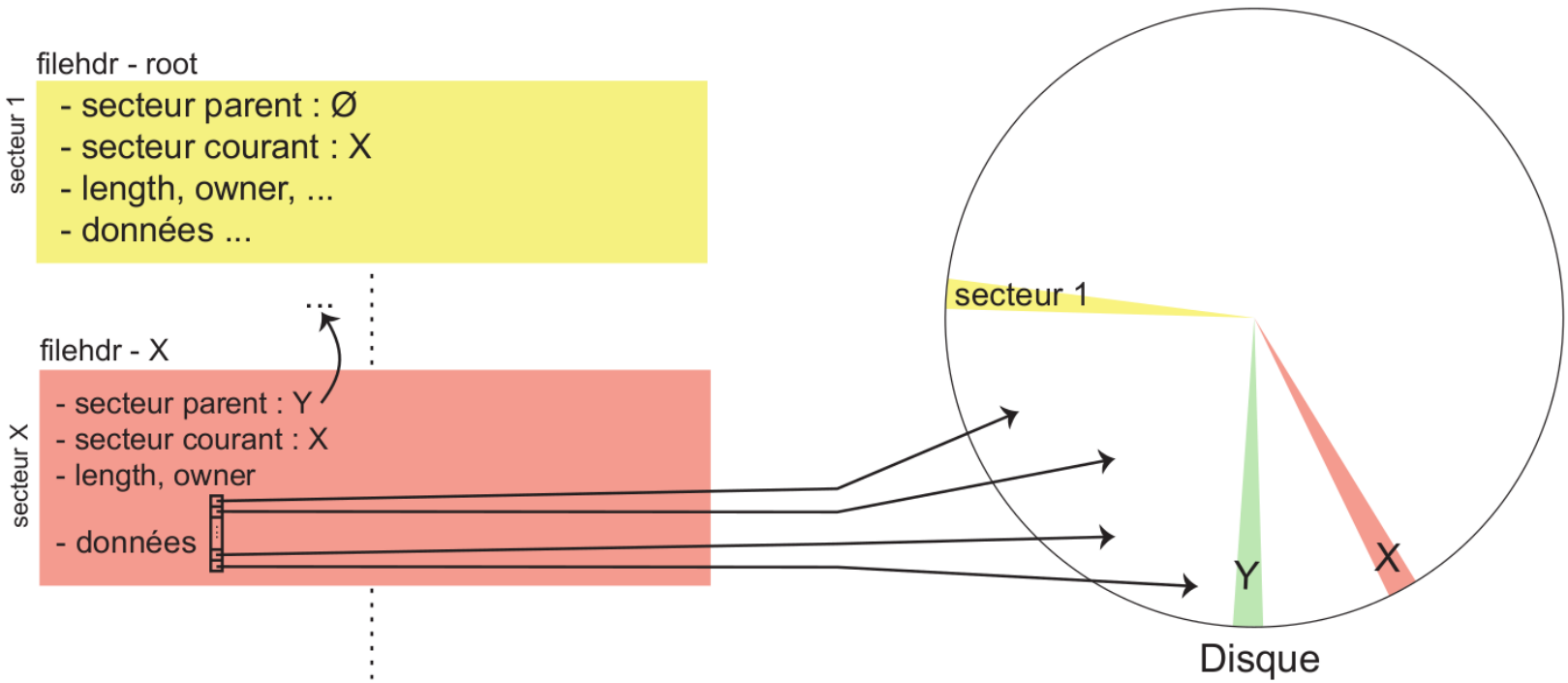
- Création d'un nouveau type utilisateur pid_t
- pid_t ForkExec(char* filename)
  - Crée un nouveau processus qui exécute le programme contenu dans le fichier filename.
  - Renvoie le pid_t correspondant au nouveau processus
- void waitpid(pid_t p)
  - Attend la terminaison du processus p
- void Proc_Exit()
  - Termine le processus

Liens de parenté entre processus



Principe

- Le secteur contenant l'i-node du répertoire Root est le 1
- Un répertoire est représenté sur disque par un fichier Nachos classique



Fonctionnalités

- Création de répertoire (mkdir)
  - Le nouveau répertoire contiendra deux fichiers : "." et ".."

Choix d'implémentation

Fonctionnalités

- Création de répertoire (mkdir)
  - Le nouveau répertoire contiendra deux fichiers : "." et ".."
- Changement de répertoire courant (cd)

Choix d'implémentation

Fonctionnalités

- Création de répertoire (mkdir)
  - Le nouveau répertoire contiendra deux fichiers : "." et ".."
- Changement de répertoire courant (cd)
  - Parcours de l'arborescence grâce aux fichiers spéciaux "." et ".."

Choix d'implémentation

Fonctionnalités

- Création de répertoire (mkdir)
  - Le nouveau répertoire contiendra deux fichiers : "." et ".."
- Changement de répertoire courant (cd)
  - Parcours de l'arborescence grâce aux fichiers spéciaux "." et ".."
  - Le répertoire racine s'appelle "/"

Choix d'implémentation

Fonctionnalités

- Création de répertoire (mkdir)
  - Le nouveau répertoire contiendra deux fichiers : "." et ".."
- Changement de répertoire courant (cd)
  - Parcours de l'arborescence grâce aux fichiers spéciaux "." et ".."
  - Le répertoire racine s'appelle "/"

Choix d'implémentation

- Le répertoire courant est commun à tout le système

Fonctionnalités

- Création de répertoire (mkdir)
  - Le nouveau répertoire contiendra deux fichiers : "." et ".."
- Changement de répertoire courant (cd)
  - Parcours de l'arborescence grâce aux fichiers spéciaux "." et ".."
  - Le répertoire racine s'appelle "/"

Choix d'implémentation

- Le répertoire courant est commun à tout le système
- Pas de chemin

Appels systèmes

- int Create (char *name, int size);
  - Retourne 1 si l'execution s'est bien passé, 0 sinon
  - Créé un fichier "name" dans le répertoire courant,de taille "size"

Appels systèmes

- int Create (char *name, int size);
  - Retourne 1 si l'execution s'est bien passé, 0 sinon
  - Créé un fichier "name" dans le répertoire courant,de taille "size"
- OpenFileId Open (char *name);
  - Ouvre le fichier "name", retourne un OpenFileId, manipulable par l'utilisateur
  - Renvoie -1 si l'ouverture est impossible

Appels systèmes

- int Write (char *buffer, int size, OpenFileId id);
  - Ecrit la chaine de taille "size" contenue dans "buffer" dans le fichier "id"
  - Renvoie le nombre de bytes effectivement écrits

Appels systèmes

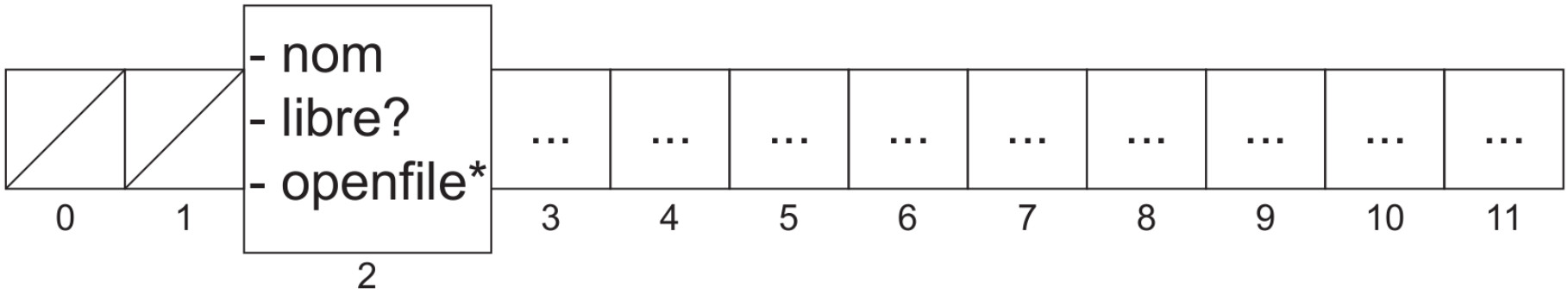
- int Write (char *buffer, int size, OpenFileId id);
  - Ecrit la chaine de taille "size" contenue dans "buffer" dans le fichier "id"
  - Renvoie le nombre de bytes effectivement écrits
- int Read (char *buffer, int size, OpenFileId id);
  - Remplis "buffer" avec un nombre de caractères "size" lus depuis le fichier "id"
  - Renvoie le nombre de bytes lus

Appels systèmes

- int Write (char *buffer, int size, OpenFileId id);
  - Ecrit la chaine de taille "size" contenue dans "buffer" dans le fichier "id"
  - Renvoie le nombre de bytes effectivement écrits
- int Read (char *buffer, int size, OpenFileId id);
  - Remplis "buffer" avec un nombre de caractères "size" lus depuis le fichier "id"
  - Renvoie le nombre de bytes lus
- void Close (OpenFileId id);
  - Ferme le fichier "id"

Table des fichiers ouverts

- Interface entre le système de fichiers et le programme utilisateur.
- Nachos permet l'ouverture de 10 fichiers simultanément au total au niveau des programmes utilisateurs
- Les indices 0 et 1 se sont jamais utilisés, ils sont réservés.
- Commune à tout le système



Projet système
Master 1

R. Blanc A.Castel C.Eymond Laritaz M.Garnier

UFR IM^{2}AG
Université Grenoble Alpes

Jeudi 17 Janvier 2019