Systèmes centralisés : TD6

- **mmap**
  - Associer un (fragment de) fichier à une zone (contiguë) de mémoire virtuelle : le contenu mémoire est une projection du (fragment de) fichier
    - Un moyen efficace de réaliser des traitements sur un fichier de façon privée, ou partagée avec d'autres processus
    - Un moyen de partager des données entre processus, en passant par un fichier mappé en mémoire dans chacun des processus
  - mmap permet aussi de réserver des pages de mémoire virtuelle sans couplage avec un fichier : ces pages peuvent être partagées entre plusieurs processus

Systèmes centralisés : TD6

void* mmap(void **addr*, size_t *length*, int *prot*, int *flags*, int *fd*, off_t *offset*);

#include <sys/mman.h**>**

- Retourne l'adresse de l'espace mémoire alloué
- Si addr = NULL ➔ adresse choisie par le noyau
- flags :
  - MAP_SHARED : projection partagée. Les modifications de la projection sont visibles par les autres processus qui projettent le fichier, et sont appliquées à ce fichier
  - MAP_PRIVATE : projection privée. Les modifications de la projection ne sont pas visibles par les autres processus qui projettent le fichier, et ne sont pas appliquées à ce fichier
  - MAP_ANONYMOUS : La projection n'est supportée par aucun fichier. Son contenu est initialisé à zéro. Les arguments *fd* (-1) et *offset* sont ignorés (0).

Systèmes centralisés : TD6

void* mmap(void **addr*, size_t *length*, int *prot*, int *flags*, int *fd*, off_t *offset*);

- length (sauf pour MAP_ANONYMOUS) : Le contenu d'une projection de fichier est initialisé avec *length* octets à partir de la position *offset* (multiple de taille de page) dans le fichier. Si length+offset > taille du fichier, les accès au-delà de la taille du fichier peuvent conduire on des signaux SIGBUS ou SIGSEGV.

- prot indique la protection mémoire :
  - PROT_NONE : aucun accès
  - Ou binaire entre :
    - PROT_EXEC : exécution de code possible
    - PROT_READ : lecture
    - PROT_WRITE : écriture

Systèmes centralisés : TD6

Exemples :

size_t pagesize = sysconf(_SC_PAGESIZE);

int fd = open ("fichier", O_RDWR); if (fd < 0) { … ; exit(1); }

char* base = mmap (NULL, pagesize, PROT_WRITE | PROT_READ, MAP_SHARED, fd ,0);

// 1ère page du fichier mappée en mémoire et partagée en Lecture/Ecriture

// Plusieurs processus qui exécutent la même séquence, peuvent faire des lectures et des écritures sur les mêmes données

size_t pagesize = sysconf(_SC_PAGESIZE);

char* base = mmap(NULL, pagesize, PROT_WRITE | PROT_READ, MAP_SHARED | MAP_ANON, -1, 0);

// une page mémoire partagée en Lecture/Ecriture

// la page peut être partagée par tous les processus qui en héritent

Systèmes centralisés : TD6

int mprotect(void ****addr*, size_t *length, int prot*);**

permet de modifier totalement ou partiellement les droits d'accès a un segment.

Les droits doivent être cohérents avec les droits d'ouverture du fichier

int munmap(void ****addr*, size_t *length*);**

- Détruit la projection mémoire
- Toute référence ultérieure déclenche une erreur

Systèmes centralisés : API en langage C

Exemple : couplage de fichier

Père

- Crée un fichier contenant 2 pages remplies de ‘a’
- Ouvre le fichier en RDWR
- Le couple en R | W, SHARED
- crée un fils
- Remplit la 2ème page avec ‘b’
- attend la fin du fils
- Lit et affiche les 10 1er octets de la 1ère page

Fils

- Attend 2 secondes
- Lit et affiche les 10 1er octets de chaque page
- Remplit la page 1 avec ‘c’

Systèmes centralisés : API en langage C

mmap_shared_file.c

```
void garnir(char zone[], int lg, char motif) {
   int ind;
   for (ind=0; ind<lg; ind++) {
      zone[ind] = motif ;
   }
}

void lister(char zone[], int lg) {
   int ind;
   for (ind=0; ind<lg; ind++) {
      printf("%c",zone[ind]);
   }
   printf("\n");
}
```

Systèmes centralisés : API en langage C

```
int main(int argc,char *argv[]) {
    int desc, wstatus;
    int taillepage = sysconf(_SC_PAGESIZE);
    char tampon[2*taillepage];
    char* base;
    if ((desc = open("temp", O_RDWR | O_CREAT |
        O_TRUNC, 0600)) == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }
    garnir(tampon, 2*taillepage, 'a');
    if ((write (desc,tampon,2*taillepage)) == -1) {
        perror("write");
    } else { printf("remplissage du fichier\n"); }
    base = mmap(NULL, 2*taillepage,
    PROT_READ | PROT_WRITE, MAP_SHARED, desc, 0);
    if (base == MAP_FAILED) { perror("mmap"); exit(1);
```

Systèmes centralisés : API en langage C

```
if ( fork() == 0 ) { /* fils*/
    sleep(2) ;
    printf("fils, page 1 :");
    lister(base,10);
    printf("fils,page 2 : ");
    lister(base+taillepage, 10);
    garnir(base,taillepage,'c');
    exit(EXIT_SUCCESS);
}
/* père */
garnir(base+taillepage,taillepage,'b');
wait(&wstatus); printf("page 1 père: ");
lister(base,10); exit(EXIT_SUCCESS); }
```

Systèmes centralisés : API en langage C

Question :

- Affichages ?
- Contenu du fichier ?
- Peut-on remplacer le fils par n'importe quel autre processus ?
- Peut-on faire la même chose sans utiliser de fichier ?

Systèmes centralisés : API en langage C

mmap_shared_anonymous.c

```
int main(int argc,char *argv[]) {
    int wstatus, taillepage;
    char* base;
    taillepage = sysconf(_SC_PAGESIZE);
    base = mmap(NULL, 2*taillepage,
    PROT_WRITE|PROT_READ, MAP_SHARED |
    MAP_ANONYMOUS, -1, 0);
    if (base == MAP_FAILED) {
       printf("map failed \n"); exit(1);
    }
    garnir(base,taillepage,'a');
    garnir(base+taillepage,taillepage,'b');
```

Systèmes centralisés : API en langage C

Exercice :

Ecrire un programme qui couple, en mode anonyme, une zone d'adressage de la taille d'une page en interdisant tout acc ès.

Provoquer l'exception de violation m émoire en tentant un acc ès en écriture dans cette zone.

En associant un traitant aux signaux SIGBUS et SIGSEGV, modi
er la protection en utilisant la primitive mprotect de fa çon a permettre l' écriture.

Systèmes centralisés : API en langage C

```
// main
Signal (SIGSEGV, traitant);
Signal (SIGBUS, traitant);

taillepage = sysconf (_SC_PAGESIZE);
base = mmap (NULL, taillepage, PROT_NONE, MAP_PRIVATE
|MAP_ANON, -1, 0);
if (base == MAP_FAILED) {
      printf("map failed \n"); exit(1);
}

printf ("première écriture : c \n");
base[0] = 'c';
printf ("lecture : %c\n", base[0]);
printf ("seconde écriture : d \n");
base[0] = 'd';
printf ("lecture : %c\n", base[0]);
```

Systèmes centralisés : API en langage C

```
void traitant (int sig) {
    printf("Traitant : signal reçu %d\n", sig);
    int ret = mprotect (base, taillepage, PROT_WRITE | PROT_READ);
    printf ("résultat mprotect : %d\n", ret);

    /* Important de mettre PROT_READ avec PROT_WRITE
    -> parce que sur certaines architectures (RISC) l'écriture est précédée
    d'une lecture, pour installer les données écrites dans le cache, et en
    permettre la gestion selon la politique du cache (LRU...) sans faire de
    cas particulier pour l'écriture.
    A noter que sur d'autres architectures (i386), mprotect positionne
    automatiquement PROT_READ, dès lors que l'on positionne
    PROT_WRITE

    -> reprise après traitement de SIGSEGV : selon les architectures,
    l'instruction ayant provoqué le SIGSEGV est reprise ou sautée.
    Si elle est reprise, et que PROT_READ n'est pas positionné, risque
    de boucler à l'infini. */
}
```

Systèmes centralisés : API en langage C

On considère le programme suivant :

```
char *base; // adresse de base de page
long pagesize; // taille d'une page
jmp buf env; // zone de sauvegarde de point de reprise

void handler (int quelsignal) { ... } // traitant du signal SIGSEGV

int main ( int argc, char *argv[]) { // programme principal
        pagesize = sysconf(_SC_PAGESIZE);
        signal(SIGSEGV,handler);
        setjmp(env);
        printf("base[0] = %c\n",base[0]);
}
```

Que devrait faire le traitant du signal pour que l'instruction printf soit bien exécutée ?

Systèmes centralisés : API en langage C

```
void handler (int quelsignal) {
      base = mmap(0, pagesize, PROT WRITE | PROT READ, MAP PRIVATE |
      MAP ANON, -1, 0);
      base[0] = 'A';
      longjmp(env,1);
 }
```

Exercice : Ecrire une impl émentation de la commande cp f1 f2 bas ée sur le couplage m émoire.

- `memcpy (dst, src, taille)` permet de copier un segment mémoire dans un autre
- pour fixer la taille d'un fichier initialement vidé

```
lseek (file, taille - 1, SEEK_SET);
write (file_out, &un_caractere, 1);
```

Systèmes centralisés : API en langage C

Exemple 3 : Copie de fichier en utilisant mmap

```
int main (int argc, char *argv[]) {
    int file_in, file_out, taille;
    char *src, *dst; char buf [2];
    struct stat statbuf; //carcateristiques du fichier
    if (argc != 3) {
        printf("usage: %s <fichier source> <fichier
        destination>\n", argv[0]);
        exit(1);
    }
    /* ouvrir et coupler fichier_source */
    if ((file_in = open (argv[1], O_RDONLY)) == -1) {
        printf("erreur ouverture source\n"); exit(2);
    }
    /* recuperer la taille de fichier_source */
    if (fstat (file_in,&statbuf) < 0){
        printf("erreur fstat"); exit(3);
    }
    taille = statbuf.st_size;
```

Systèmes centralisés : API en langage C

```
/* projection memoire en prive */
if ((src = mmap (NULL, taille, PROT_READ, MAP_PRIVATE,
file_in, 0)) == (caddr_t) -1) {
   printf("erreur couplage source\n"); exit(4); }

/* ouvrir et coupler destination */
if ((file_out = open (argv[2], O_RDWR | O_CREAT |
O_TRUNC, 0644)) == -1) {
/* O_RDWR necessaire car mmap impose qu'un fichier
couplé soit (au moins) ouvert en lecture */
   printf("erreur ouverture destination\n"); exit(5);
}

/* Il faut fixer la taille du fichier destination à la
taille du fichier source avant le couplage. lseek
(taille du fichier source) + write d'un octet*/
buf[0]='x';
lseek (file_out, taille - 1, SEEK_SET);
write (file_out, buf, 1);
```

Systèmes centralisés : API en langage C

```
if ((dst = mmap (NULL, taille, PROT_WRITE,
MAP_SHARED, file_out, 0)) == (caddr_t) -1) {
    /* MAP_SHARED est necessaire pour que les
    ecritures soient visibles dans le fichier */
    printf("erreur couplage destination\n");
    exit(6);
}
memcpy (dst, src, taille); close(file_in);
close(file_out);
exit(0);
}
```