1 TD1

1.1 Ex1

P1 à t = 0	P2 à t = 1
Calcul pendant 3 unités	Calcul pendant 2 unités
Impression sur imprimante pendant 2 unités	Affichage à l'écran pendant 2 unités
Calcul pendant 2 unités	Calcul pendant 1 unité
Affichage à l'écran pendant 1 unité	Impression sur imprimante pendant 2 unités
Calcul pendant 3 unités	FIN.
FIN.	

1) Mono-programmé

CPU	p_1	p_1	p_1			p_1	p_1		p_1	p_1	p_1	p_2	p_2			p_2		
Impression				p_1	p_1												p_2	p_2
Écran								p_1						p_2	p_2			
Attente		p_2	p_2	p_2	p_2	p_2	p_2	p_2	p_2	p_2	p_2							

Occupation Processeur $\frac{11}{18} \rightarrow 61\%$

2) Multi-programmé

CPU	p_1	p_1	p_1	p_2	p_2	p_1	p_1	p_2	p_1	p_1	p_1
Impression				p_1	p_1				p_2	p_2	
Écran						p_2	p_2	p_1			
Attente		p_2	p_2								

Occupation Processeur $\frac{11}{11} \rightarrow 100\%$

3) Temps Partagé (1s)

CPU	p_1	p_2	p_1	p_2	p_1		p_2	p_1	p_1		p_1	p_1	p_1
Impression						p_1	p_1	p_2	p_2				
Écran					p_2	p_2				p_1			
Attente		p_1	p_2	p_2									

Occupation Processeur $\frac{11}{13} \rightarrow 84\%$

1.2 Ex2

1.2.1 États possibles de processus

- Prêt : Attente du processeur
- En Exécution :
- Bloqué : Attente IO

1.2.2 Files d'attente nécessaires pour gérer les processus de ce système.

Plusieurs programmes coexistent à l'état d'attente (ou bloqué) → deux files

1.2.3 Comment gérer les priorités

Favoriser les processus demandant de l'I/O en augmentant la priorité du processus lors de l'accès à l'I/O (puis de le réduire si il n'en a pas fait depuis un certain temps)

1.2.4 PCB

— PID
— État
  · AX, BX, CX, DX, EX
  · CO
  · FLAGS
  · DS, CS
— Cause du blocage
— Priorité, quantum

1.3 Ex3

On représente RM avec

EX	**A**	**B**	**C**	**D**	**H**
x	x	x	x	x	x

Chaque interruption à un masque propre afin de représenter la priorité :

EX	0	0	0	0	0	0
A	1	0	0	0	0	0
B	1	1	0	0	0	0
C	1	1	1	0	0	0
D	1	1	1	1	0	0
H	1	1	1	1	1	0

1.4 Ex4

Exemple foireux et inutile.

2 TD2

2.1 Ex1

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <time.h>

pid_t chld;

void usr1Handler() {
    printf(" Temperature %d\n", (rand() % 30) + 10);    fflush(stdout);
}

void alrmHandler() {
    kill(chld, SIGUSR1);    alarm(5);
}

int main(int argn, char* argv[]) {
    struct sigaction newUsr1Handler, newAlrmHandler;
    newUsr1Handler.sa_handler = usr1Handler;
    newAlrmHandler.sa_handler = alrmHandler;

    srand(time(NULL));

    if ((chld = fork()) == -1) {
        perror("fork");
        return -1;
    }

    /* Child's Process */
    if(chld == 0) {
        sigaction(SIGUSR1, &newUsr1Handler, NULL);
        while(1) pause();
    }

    /* Parent's Process */
    else {
        sigaction(SIGALRM, &newAlrmHandler, NULL);
        alarm(5);

        while(1) {
            sleep(1);   putchar('-');   fflush(stdout);
        }
    }

    return 0;
}
```

2.2 Ex2

```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <time.h>

pid_t chld;
char curChar;
char step = 1;

void child_handler() {
    for(int i= 0; i< step && curChar <= 'z'; i++) {
        printf("%c", curChar);
        curChar++;
    }
    step++;
    kill(getppid(), SIGUSR1);
}

void parent_handler() {
    for(int i= 0; i< step && curChar <= 'Z'; i++) {
        printf("%c", curChar);
        curChar++;
    }
    if(curChar <= 'Z') {
        step++;
        kill(chld, SIGUSR1);
    }
}

int main(int argn, char* argv[]) {
    struct sigaction sa;

    if ((chld = fork()) == -1) {
        perror("fork");
        return -1;
    }

    /* Child's Process */
    if(chld == 0) {
        sa.sa_handler = child_handler;
        curChar = 'a';
        sigaction(SIGUSR1, &chldUsr1Handler, NULL);
        while(1) pause();
    }

    /* Parent's Process */
    else {
        sa.sa_handler = parent_handler;
        curChar = 'A';
        sigaction(SIGUSR1, &prntUsr1Handler, NULL);
        kill(chld, SIGUSR1);
        while(1) pause();
    }

    return 0;
}
```

3 TD3

3.1 Ex1 – Programmation d'un Robot

Un robot dispose de 2 roues de circonférence 10cm. Pour faire avancer le robot on fait appel au SVC (appel système). Le contrôleur du robot dispose de deux registres :

— RE : Registre d'état mis à 1 quand le robot est prêt.
— RC : Registre faisant avancer le robot d'un tour de roue quand mis à 1.

3.1.1 Sans mécanisme d'interruption – Mono-processus

```
def rouler(n):
    save_context(PCB[actif])

    for i in range(n):
        while RE != 1
            wait()
        RC = 1

    load_context(PCB[actif])    ;
       load_psw(PCB[actif].psw)
```

3.1.2 Avec mécanisme d'interruption – Multi-processus

En admettant qu'une interruption est envoyée quand les roues ont fait un tour complet :

```
def rouler(n):
    save_context(PCB[active])   ;   blocked_queue.push(active)

    p = packet_io(pid = PCB[active].pid, nb = n, type = 1)
    request_queue.push(p)
    new_pr = ready_queue.pop()

    load_context(PCB[active])   ;   load_psw(PCB[active].psw)

def driver_robot():
    while True:
        if not request_queue.empty:
            p = request_queue.pop()
            while(RE != 1):
                wait()
            RCC = p.type
            p.nb--
            pause()
            ready_queue.push(blocked_queue.pop())

def robot_int_routine(p):
    save_context(PCB[active]);
    if p.nb = 0:
        # envoie d'un signal pour reveiller le driver
    else:
        RC = p.type
        p.nb--

    load_context(PCB[active])   ;   load_psw(PCB[active].psw)
```

3.2 Ex2

Un disque comporte 200 pistes ([0..199]). La tête de lecture est en 112. File d'attente :

138 91 165 67 158 43 132 28 106 84

3.2.1 Durée de déplacement par algos

FIFO	First-in First-out	732
PCTR	Plus court temps de recherche	202
SCAN	Va d'un bout à l'autre en répondant au passage puis demi tour	258
LOOK	Pareil que SCAN mais va pas au bout si inutile	190

3.2.2 Danger d'autoriser traitement immédiat nouvelles requêtes ? Quels algorithmes sont sujets à ces famines ?

Peut stagner si toutes les opérations se font au même endroit, oubliant les autres. Le PCTR.

3.3 Ex3

Un fichier occupant 100 blocs et un répértoire ou un index occupant un bloc, combien de transferts disque-ram sont nécessaires pour ajouter un bloc au début du fichier et enlever un bloc à la fin du fichier :

Contiguë

—

—

Chaînage

— Accès au répertoire, écriture sur le disque : 2

— Accès au répertoire, parcours de tous les blocs : 101

Chaînage Indexé

— Accès au répertoire, écriture sur le disque : 2

— Accès au répertoire : 1

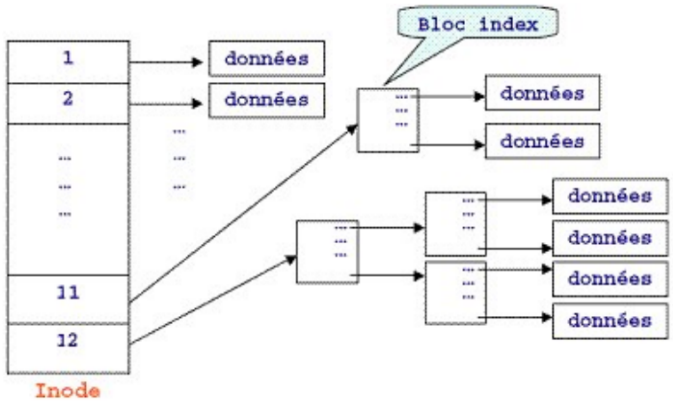
Indexé

— Accès au répertoire, écriture sur le disque : 2

— Accès au répertoire : 1

3.4 Ex4

Avec un i-node comme ça :



Chaque bloc d'index pouvant contenir 256 adresse de blocs, combien d'accès sont nécessaires pour accéder au 583ème bloc d'un fichier qui en compte 896 ?

4 TD4

4.1 Ex1

4.1.1

Fonction copyFile(int f1, int f2) copiant le contenu d'un fichier dans un autre et retournant le nombre d'octets copiés. La copie se fait par blocs de 1024o. Programme affichant deux fichiers puis la taille lue grâce à deux processus parent et fils :

```c
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <unistd.h>

#define READSIZE 1024

int copyFile(int f1, int f2) {
    int totalByteRead= 0, lastByteRead= READSIZE;
    char buffer[READSIZE];

    while(lastByteRead == READSIZE) {
        lastByteRead = read(f1, buffer, READSIZE);
        write(f2, buffer, lastByteRead);
        totalByteRead += lastByteRead;
    }

    return totalByteRead;
}

int main(int argc, char* argv[]) {
    int file;
    pid_t chld = fork();

    if(chld == 0)   { file = open((argc > 2) ? argv[2] : "file2",
       O_RDONLY);             }
    else            { file = open((argc > 1) ? argv[1] : "file1",
       O_RDONLY); wait(NULL); }

    printf("\n%d bytes\n", copyFile(file, fileno(stdout)));
    close(file);
    return 0;
}
```

4.1.2

Pareil mais en utilisant une mémoire partagée pour que le fils indique au père son nombre d'octets lus :

```
int main(int argc, char* argv[]) {
    int file;
    id_t id = shmget(IPC_PRIVATE, sizeof(int), 0666);
    int* shmArea = (int*) shmat(id, NULL, 0);

    pid_t chld = fork();

    if(chld == 0)   file = open((argc > 2) ? argv[2] : "file2", O_RDONLY);
    else            file = open((argc > 1) ? argv[1] : "file1", O_RDONLY);

    int byteCopied = copyFile(file, fileno(stdout));
    close(file);

    if(chld == 0) *shmArea = byteCopied;
    else
    {
        wait(NULL);
        printf("Bytes copied: %d & %d\n", byteCopied, *shmArea);

        shmdt((void*) shmArea);
        shmctl(id, IPC_RMID, NULL);
    }

    return 0;
}
```

4.1.3

```
#include <unistd.h>

int main(int, char**) {
    int pid, pip[2];
    char instring[20];
    pipe(pip);
    pid = fork();

    // Child
    if (pid == 0) {
        close(pip[0]);
        write(pip[1], "Salut !", 7);
    }

    // Parent
    else {
        close(pip[1]);
        read(pip[0], instring, 7);
    }

    return 0;
}
```

4.2 Ex2

Application serveur – client échangeant a-z :

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#define SHMSZ 27

void server() {
    char c;
    int shmid;
    key_t key = 5678;
    char *shm, *s;

    // Creer le segment
    shmid = shmget(key, SHMSZ, IPC_CREAT | 0666);
    if(shmid < 0) { perror("shmget"); exit(1); }

    // Attacher le segment
    shm = shmat(shmid, NULL, 0);
    if (shm == (char *) -1) { perror("shmat"); exit(1); }

    // Mettre quelques choses dans la memoire pour l'autre processus
    s = shm;
    for (c = 'a'; c <= 'z'; c++) *s++ = c;
    *s = NULL;

    // On attend que le client lise en mettant en premier caractere '*'
    while (*shm != '*') sleep(1);

    shmdt((void*) shm);
    shmctl(shmid, IPC_RMID, NULL);
    exit(0);
}

void client() {
    int shmid;
    char *shm, *s;

    // Obtenir le segment "5678" cree par le serveur
    key_t key = 5678;
    shmid = shmget(key, SHMSZ, 0666);
    if(shmid < 0) { perror("shmget"); exit(1); }

    // Attacher le segment a notre espace de donnees
    shm = shmat(shmid, NULL, 0)
    if (shm == (char*) -1) { perror("shmat"); exit(1); }

    // Lire ce que le serveur a mis dans la memoire
    for (s = shm; *s != NULL; s++) putchar(*s);
    putchar('\n');

    // Changez le premier caractere du segment en '*' pour indiquer la
       lecture du segment
    *shm = '*';
    exit(0);
}
```

4.3 Ex3

Programmes modifiant et affichant un fichier binaire composé de 10 entiers à l'aide d'une mémoire partagée :

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/mman.h>
#define FILESIZE 10

int main(int argc, char* argv[]) {
    int i= 0, fd = open("titi.dat", O_RDWR, 0666);

    int* fileMap = (int*) mmap(NULL, FILESIZE*sizeof(int), PROT_READ |
       PROT_WRITE, MAP_SHARED, fd, 0);

    if(fileMap == (int*) MAP_FAILED) { perror("mmap"); exit(1); }
    close(fd);

    while(1) {
        scanf("%d", &i);
        if(i == 99) break;
        if(i < 10 && i >= 0) fileMap[i] = fileMap[i]+1;
    }

    munmap((void*) fileMap, FILESIZE*sizeof(int));
    return 0;
}
```

```c
int main(int argc, char* argv[]) {
    int i= 0, fd = open("titi.dat", O_RDONLY, 0666);

    int* fileMap = (int*) mmap(NULL, FILESIZE*sizeof(int), PROT_READ,
       MAP_PRIVATE, fd, 0);

    if(fileMap == (int*) MAP_FAILED) { perror("mmap"); exit(1); }
    close(fd);

    while(1) {
        scanf("%d", &i);
        if(i == 99) break;
        for(int j= 0; j< 10; j++) printf("\t%d\n", fileMap[j]);
    }

    munmap((void*) fileMap, FILESIZE*sizeof(int));
    return 0;
}
```

4.4 Ex4

Intervertir les caractères d'un fichier :

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/mman.h>

int main(int argc, char* argv[]) {
    int i= 0, fd = open(filename, O_RDWR, 0666);
    char* filename = "file.txt";
    struct stat st;

    stat(filename, &st);
    long fileSize = st.st_size;
    char* fileMap = (char*) mmap(NULL, fileSize, PROT_READ | PROT_WRITE,
       MAP_SHARED, fd, 0);

    if(fileMap == (char*) MAP_FAILED) { perror("mmap"); exit(1); }
    close(fd);

    while(i < fileSize/2) {
        char c = fileMap[i];
        fileMap[i] = fileMap[fileSize - i-1];
        fileMap[fileSize - i-1] = c;
        i++;
    }

    printf("%s\n", fileMap);
    munmap((void*) fileMap, fileSize);
    return 0;
}
```