LEPL1503/LSINC1503 - Cours 4

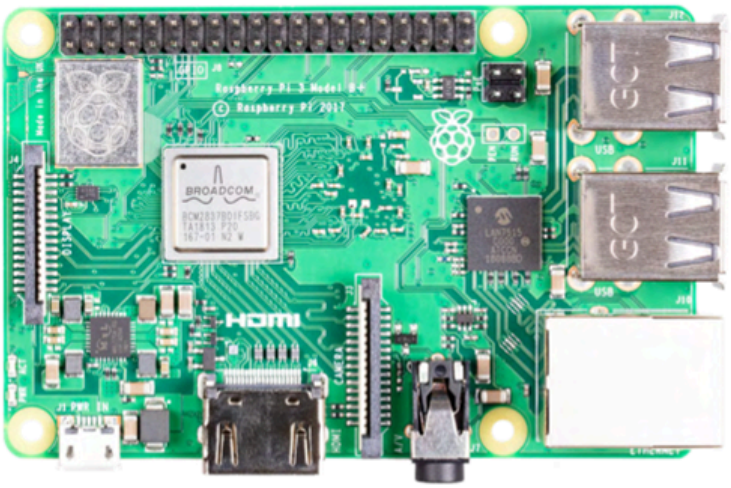
O. Bonaventure, B. Legat, M. Baerts

☐ Full Width Mode ☐ Present Mode

Table of Contents

Ordinateurs actuels

Votre Raspberry pi 3B+



Source

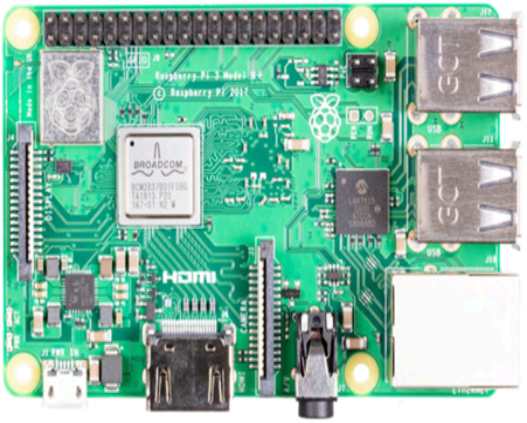
Raspberry Pi 3 Model B+ has a 64-bit quad-core processor running at 1.4GHz, dual-band 2.4GHz and 5GHz wireless LAN, Bluetooth 4.2/BLE, Gigabit Ethernet over USB 2.0, and PoE capability via a separate PoE HAT. The dual-band wireless LAN comes with modular compliance certification.

Caractéristiques de votre Raspberry pi 3B+

1.4 GHz

2300 MIPS @ 1Ghz

Mémoire cache

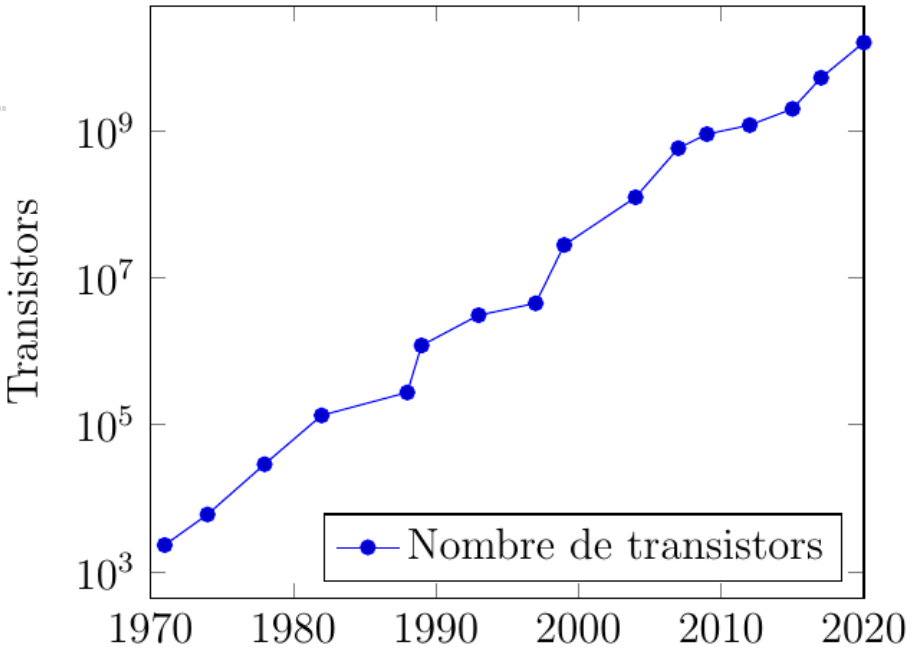


- 16 KB de cache d'instruction L1 pour chaque cœur
- 16 KB de cache de données L1 pour chaque cœur
- 512KB de cache unifiée L2

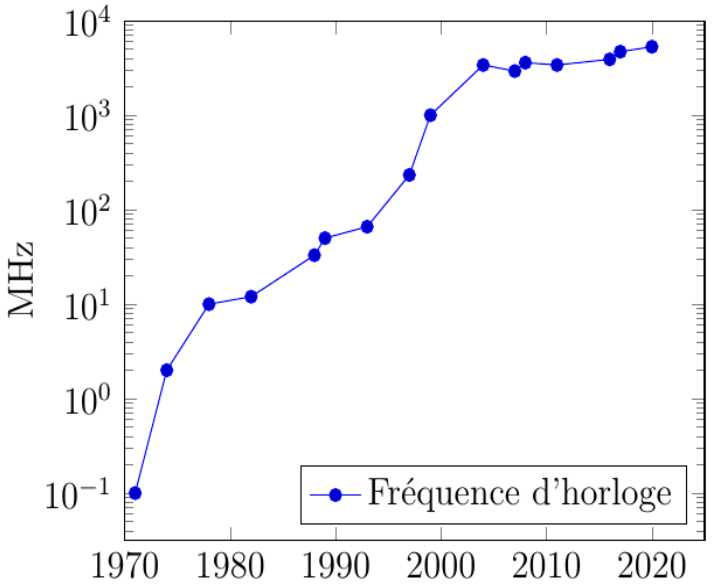
Complexité des processeurs

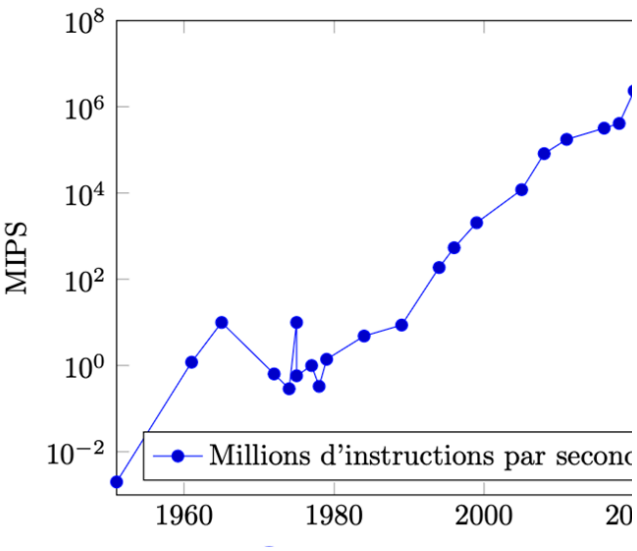
Loi de Moore

Le nombre de transistors dans un circuit intégré double quasiment tous les 2 ans

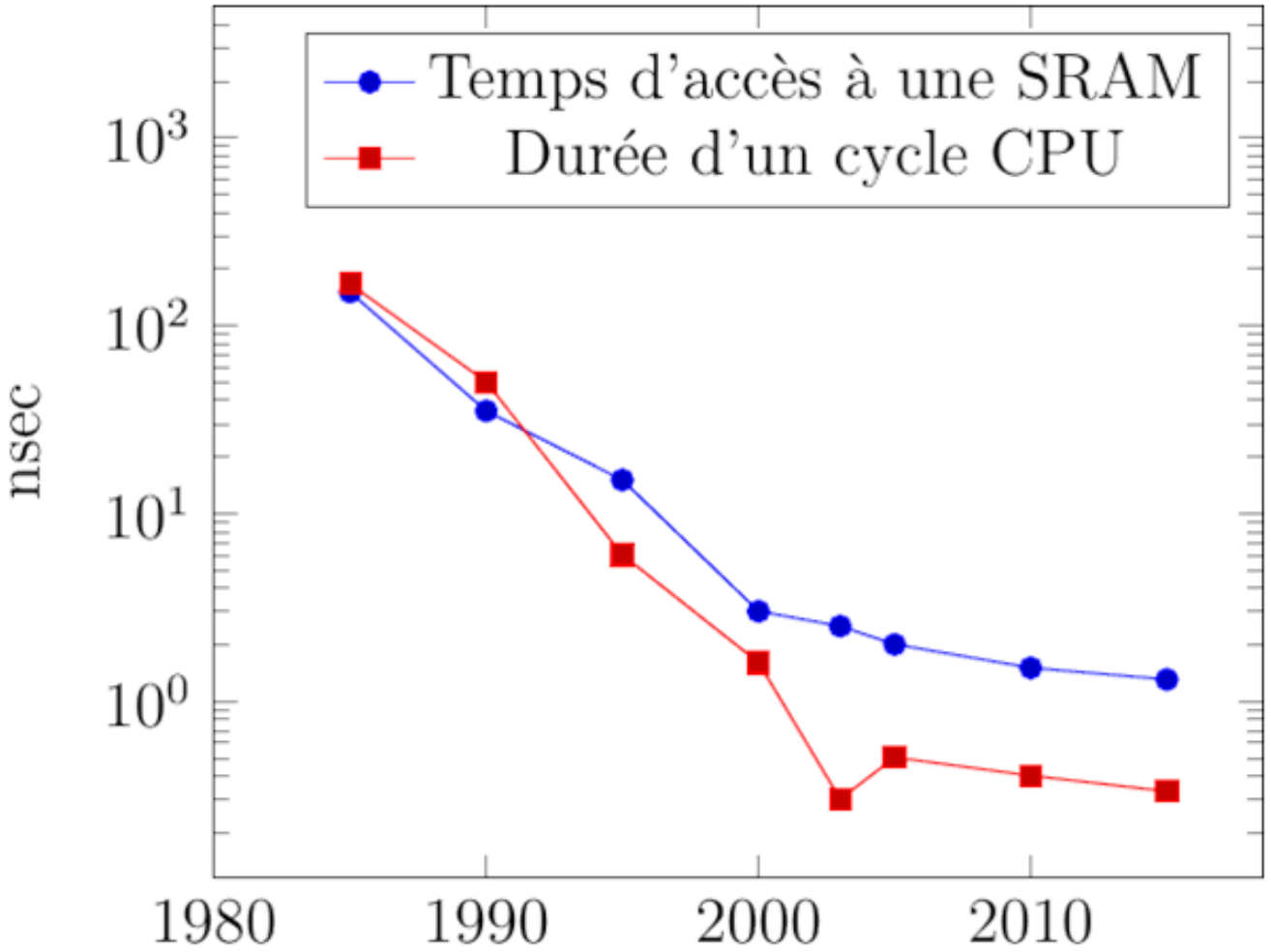


Caractéristiques des processeurs

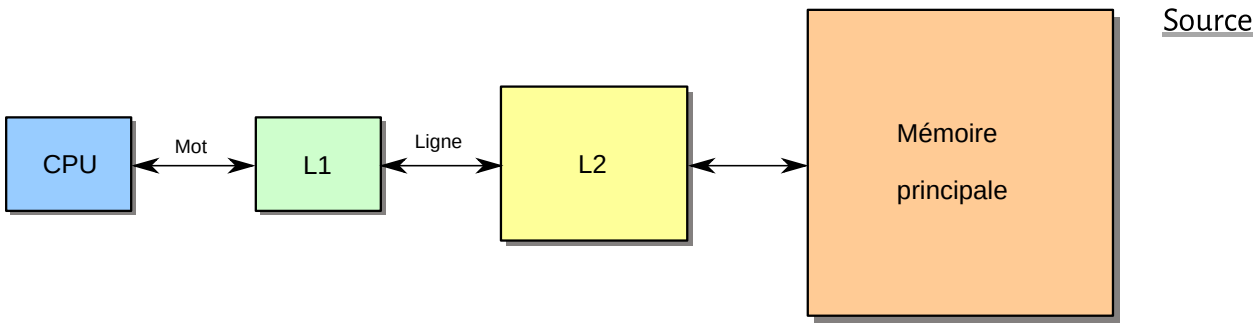




L'accès à la mémoire



Les mémoires cache



Source

Exemple : multiplication de matrices

```
void multiply_matrices(int mat1[N]
[N], int mat2[N][N], int res[N][N])
{
int i, j, k;
for (i = 0; i < N; i++)
    {
    for (j = 0; j < N; j++)
        {
        res[i][j] = 0;
        for (k = 0; k < N; k++)
            {
            res[i][j] += mat1[i][k] *
mat2[k][j];
            }
        }
    }
}
```

```
void multiply_matrices2(int mat1[N]
[N], int mat2[N][N], int res[N][N])
{
int i, j, k;
for (j = 0; j < N; j++)
    {
    for (i = 0; i < N; i++)
        {
        res[i][j] = 0;
        for (k = 0; k < N; k++)
            {
            res[i][j] += mat1[i][k] *
mat2[k][j];
            }
        }
    }
}
```

wooclap https://app.wooclap.com/QEAHMX

▶ **Un des codes est-il plus rapide ?**

Fichiers

Les appels system

```
    int open(const char *pathname, int flags, mode_t mode); // Ouvre le fichier e
t retourne un file descriptor, en cas d'erreur return -1
    // Toutes les fonctions suivantes utilisent le file descriptor retourné par o
pen
    ssize_t read(int fd, void buf[.count], size_t count); // Permet de lire dans
un fichier, en cas d'erreur return -1
    ssize_t write(int fd, const void buf[.count], size_t count); // Permet d'écri
re dans un fichier, en cas d'erreur return -1
    off_t lseek(int fd, off_t offset, int whence); // Permet de se déplacer dans
un fichier, en cas d'erreur return -1
```

Appel system Open

```
int open(const char *pathname, int flags, mode_t mode);
Descriptions des arguments :
1. Le chemin vers le fichier
2. Le drapeau qui designe le mode d'accès du fichier
   - O_RDONLY         open for reading only
   - O_WRONLY         open for writing only
   - O_RDWR           open for reading and writing
   - O_APPEND         append on each write
   - O_CREAT          create file if it does not exist
   - O_TRUNC          truncate size to 0
3. Le mode qui représente les permissions (uniquement pour la création de fichier)
   - S_IRWXU  00700 user (file owner) has read, write, and execute permission
   - S_IRUSR  00400 user has read permission
   - S_IWUSR  00200 user has write permission
   - S_IRWXG  00070 group has read, write, and execute permission
   - S_IRGRP  00040 group has read permission
   - S_IWOTH  00002 others have write permission
```

wooclap https://app.wooclap.com/QEAHMX

▶ **Quel 2ème argument pour ouvrir un fichier en écriture, le créer si il n'existe pas et se mettre à la fin pour rajouter du contenu ?**

wooclap https://app.wooclap.com/QEAHMX

▶ **Quel 3ème argument pour donner les permissions rw-r-r- à un fichier**

Detection d'erreurs lors des appels system

Un appel système peut échouer pour diverses raisons, il faut donc vérifier le retour de la fonction appelée

- La variable globale *errno* mise à jour par **Unix** vous donne un code qui vous informe sur la nature de l'échec de l'appel de la fonction.
- Cette variable se met à jour seulement en cas d'erreur et n'est pas re-initialisée
- La fonction *void perror(const char *s)* permet d'interpreter l'erreur courante de errno
- La fonction *char *strerror(int errnum)* permet d'nterpreter l'erreur courante de errno

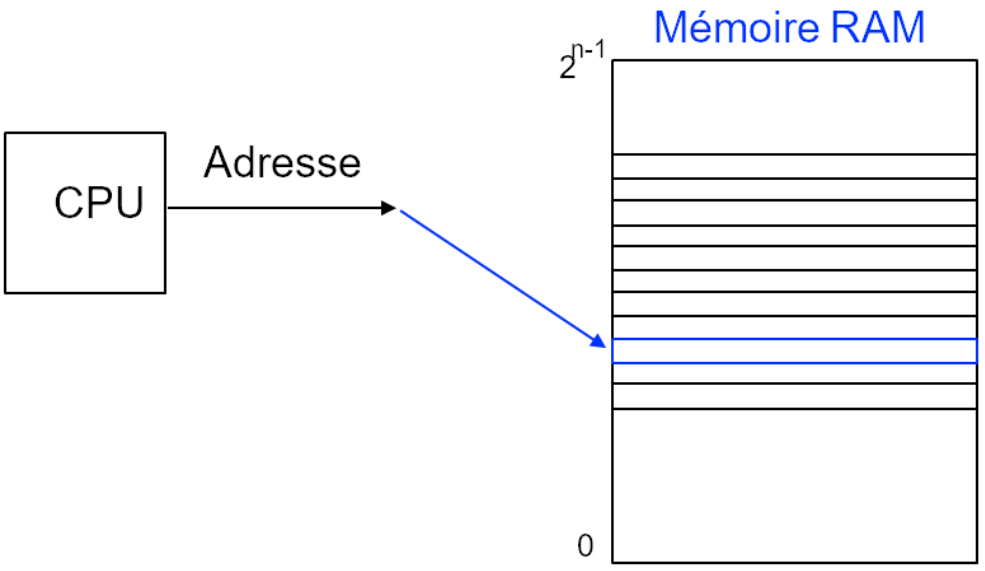
```
int fd = open(filename, flags, mode);
if (fd == -1) {
  if (errno == ENOENT) {
    // No such file or directory
  } else if (errno == EACCES)
    // Permission denied
  } else {
    perror('open');
  }
}
```

Les descripteurs de fichiers

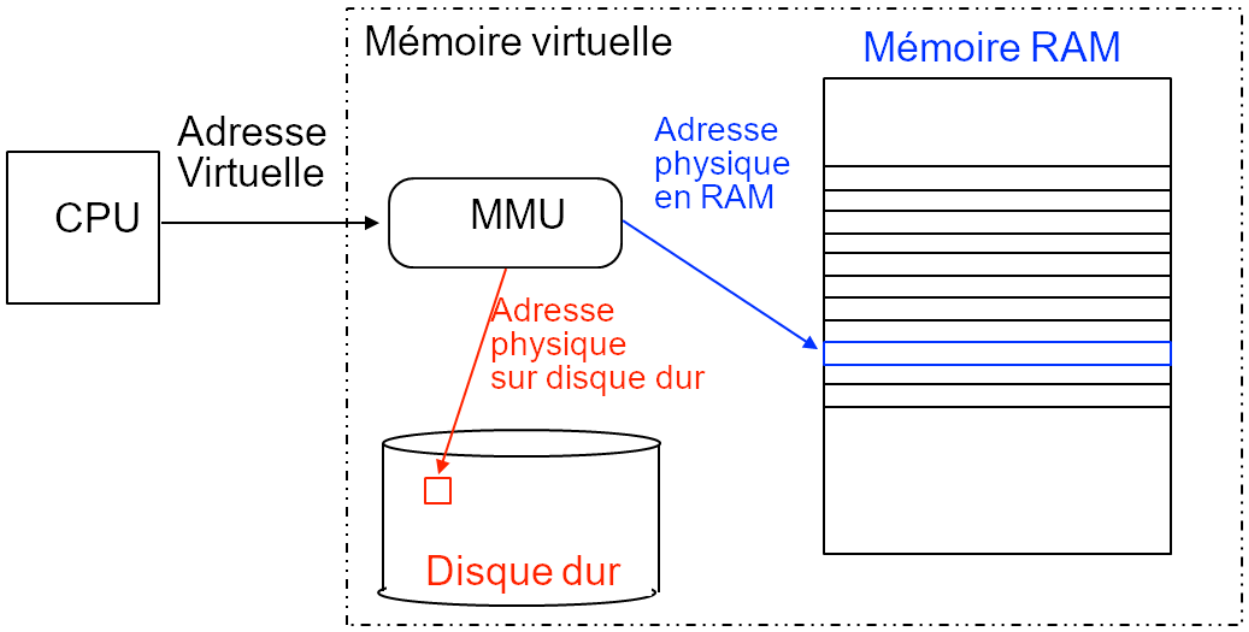
```
- Il s'agit d'un ID qui permet d'identifier les fichiers ouverts par un processus
- La liste de ces descripteurs de fichiers sont maintenus par le Noyau pour chaque
processus

| File Descriptor | Flux       |
|-----------------|------------|
| 0               | Stdin      |
| 1               | Stdout     |
| 2               | Stderr     |
| 3               | First File |
- Le file descriptor est le premier entier disponible dans la liste sinon on en cré
e un nouveau
```

```c
#include <fcntl.h> // pour open
#include <stdio.h>

int main(int argc, char **argv) {
    int fd1,fd2,fd3,err;

    fd1=open("/dev/zero",O_RDONLY);
    printf("Filedescriptor : %d\n",fd1);
    fd2=open("/dev/zero",O_RDONLY);
    printf("Filedescriptor : %d\n",fd2);
    err=close(fd1);
    fd3=open("/dev/zero",O_RDONLY);
    printf("Filedescriptor : %d\n",fd3);
}
```

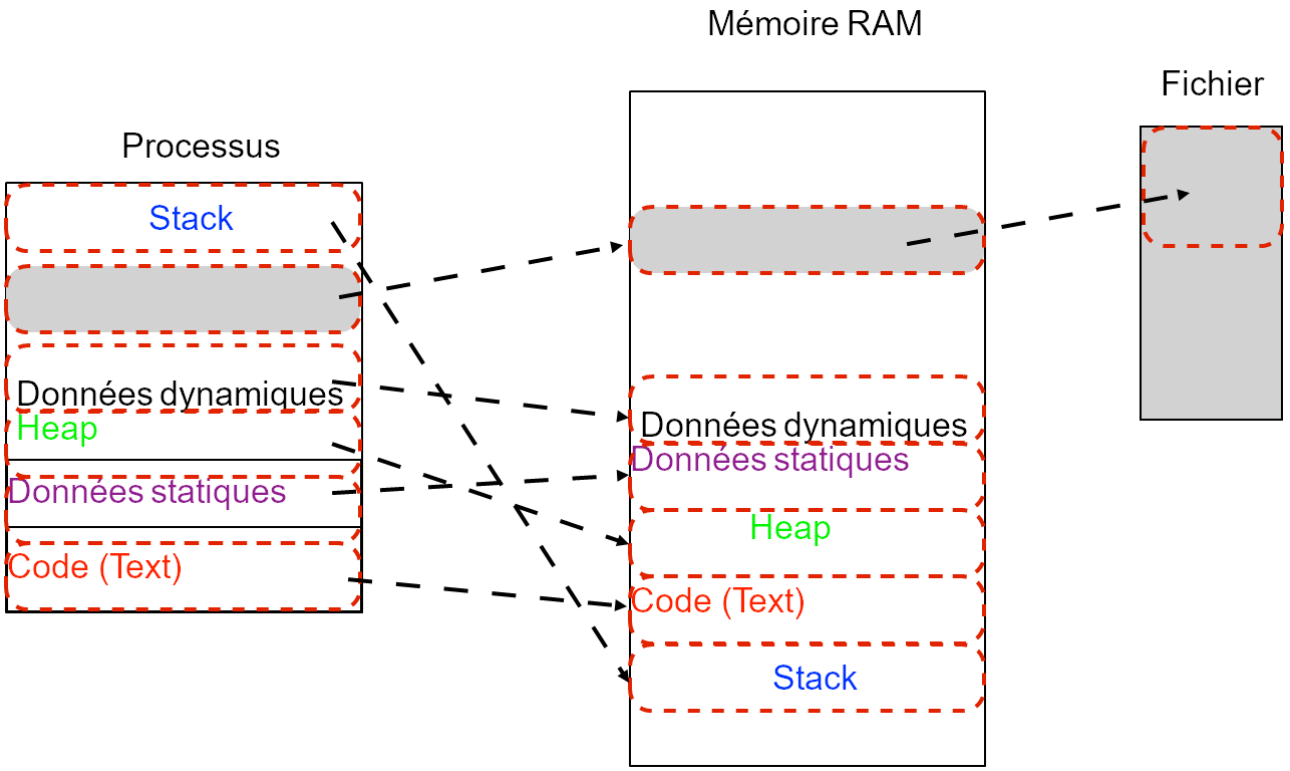
La gestion de la mémoire dans le passé



La gestion de la mémoire de nos jours



Le mappage en mémoire



Appel système mmap et munmap

```
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
Descriptions des arguments :
1. L'addresse où le le fichier a été mappé. Quand il est à NULL, le noyau choisit
l'addresse
2. La taille du fichier en octets
3. La protection appliquée à l'emplacement en mémoire
   - PROT_EXEC  Pages may be executed.
   - PROT_READ  Pages may be read.
   - PROT_WRITE Pages may be written.
   - PROT_NONE  Pages may not be accessed.
4. Les drapeaux qui determine si les mise à jour du mapping seront exclusive ou non
aux processus.
   - MAP_SHARED  Share this mapping.
   - MAP_PRIVATE Create a private copy-on-write mapping.
5. Le file descriptor obtenu avec open
6. offset permet de spécifier où il faut commencer à lire
```

```
int munmap(void addr, size_t length);
Il supprime le mapping de la mémoire
```

Threads

Les pointeurs vers les fonctions

C (C17 + GNU extensions)

```
 1  #include <stdio.h>
 2  int mul(int a, int b)
 3  {
 4    return a*b;
 5  }
 6
 7  int sum(int a, int b)
 8  {
 9    return a+b;
10  }
11
12  int main(int argc, char **argv){
13    int (*f) (int, int);
14    f = &sum;
15    printf("%d, %p\\n", f(4, 5), f)
16    f = &mul;
17    printf("%d, %p\\n", f(4, 5), f)
18
```

Edit Code & Get AI Help

line that just executed
next line to execute

< Prev Next >

Step 1 of 12

Visualized with pythontutor.com

Print output (drag lower right corner to resize)

Stack Heap

main

argc	int ?
argv	pointer to char* ?
f	pointer to function ?

Note: ? refers to an uninitialized value

C/C++ details: none [default view]

#include <stdlib.h>

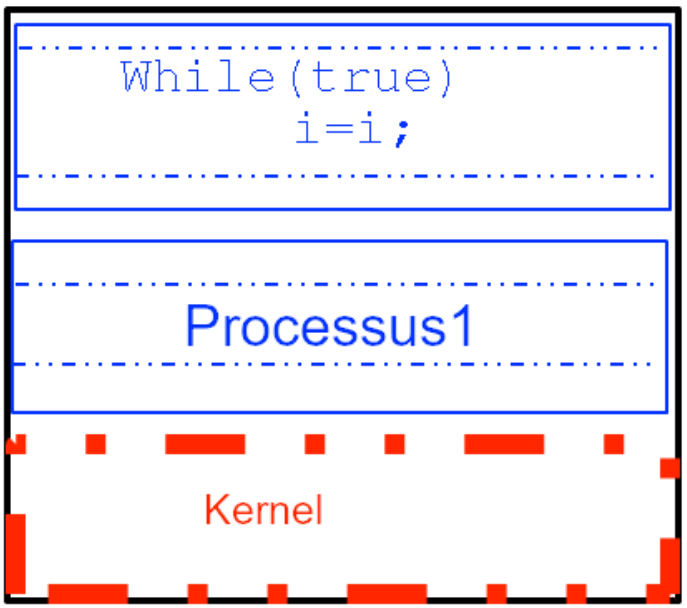
void qsort(void *base*, *sizet nmemb*, *sizet size*, *int*(compar)(const void *, const void *));

DESCRIPTION The qsort() function sorts an array with nmemb elements of size size. The base argument points to the start of the array. The contents of the array are sorted in ascending order according to a comparison function pointed to by compar, which is called with two arguments that point to the objects being compared. The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is undefined.

```
qsort(array,SIZE,sizeof(double),cmp);
int cmp(const void *ptr1, const void *ptr2) {
  const double *a=ptr1;
  const double *b=ptr2;
  if(*a==*b)
    return 0;
  else
    if(*a<*b)
      return -1;
    else
      return +1;
}
```

Partage du CPU

```
- Plusieurs processus tournent en même temps sur un ordinateur
- Exécution OS
- Hardware génère une interruption toutes t millisecondes, horloge appel système
- Un processus peut être interrompu à tout instant par le kernel !
```



Threads

```
- De nombreuses applications font plusieurs actions simultannées
- un outil graphique avec fenêtres ouvertes
- un make compilant plusieurs fichiers C
- Il est naturel de séparer ces activités, mais créer un processus pour chacune est coûteux et complique les interactions entre sous-processus
```

Les appels system Threads

```
    int pthread_create(pthread_t *restrict thread,
                       const pthread_attr_t *restrict attr,
                       typeof(void *(void *)) *start_routine,
                       void *restrict arg); // Cette fonction permet de créer le thread
    void pthread_exit(void *retval); // Cette fonction permet de terminer le thread et de retourner une valeur
    int pthread_join(pthread_t thread, void **retval); // Cette fonction demande d'attendre que le thread se finisse
```

Appel system pthread_create

```
int pthread_create(pthread_t *restrict thread,
                   const pthread_attr_t *restrict attr,
                   typeof(void *(void *)) *start_routine,
                   void *restrict arg);
Descriptions des arguments :
1. L'ID du thread
2. Les attributs du thread, il peut être NULL
3. Le foncteur qui va être exécuté
4. L'argument du foncteur
```

```
void *f1(void *param) { }
void f2(void *param) { }
void *f3(void param) { }
void f4() { }
```

wooclap https://app.wooclap.com/QEAHMX

▶ **Laquelle des fonctions suivantes**

```c
#include <pthread.h>
#include <stdio.h>
int global = 0;
void *thread_first(void * param) {
    global++;
    return(NULL); // Une facon de terminer un thread
}
void *thread_second(void * param) {
    global++;
    pthread_exit(NULL); // Une autre facon de terminer un thread
}

int main (int argc, char *argv[])  {
    pthread_t first, second;
    int err;

    err=pthread_create(&first,NULL,&thread_first,NULL);
    if(err!=0)
        perror("pthread_create");

    err=pthread_create(&second,NULL,&thread_second,NULL);

    for(int i=0; i<1000000000;i++) { /*...*/ }

    err=pthread_join(first,NULL);
        if(err!=0)
            perror("pthread_join");
    err=pthread_join(second ,NULL);
        if(err!=0)
            perror("pthread_join");
}
```

```c
pthread_t threads[NTHREADS];
int arg[NTHREADS];
void *neg (void * param) {
    int *l;
    l=(int *) param;
    int r=-*l;
    return ((void *) &r);
}
int main (int argc, char *argv[])  {
    int err;
    for(long i=0;i<NTHREADS;i++) {
        arg[i]=i;
        err=pthread_create(&(threads[i]),NULL,&neg,(void *) &(arg[i]));
        if(err!=0)
        error(err,"pthread_create");
        int *r;
        err=pthread_join(threads[i],(void **)&r);
        printf("Resultat[%d]=%d\n",i,*r);
        if(err!=0)
        error(err,"pthread_join");
    }
}
```

wooclap https://app.wooclap.com/QEAHMX

▶ **Le code est-il correct ?**

```c
pthread_t t1;

void *f(void *param) { }
void launch(void ){
pthread_t t2;
pthread_t * t3=(pthread_t *) malloc(sizeof(pthread_t));

int err=pthread_create(&t1,NULL,&f, v);
err=pthread_create(t1,NULL,&f, v);
err=pthread_create(&t2,NULL,&f, v);
err=pthread_create(t3,NULL,&f, v);
}
```

wooclap https://app.wooclap.com/QEAHMX

▶ **Lequel ou lesquels des appels a pthread_create est correct**

```c
int fd1;

void *f1(void *param) {
  char buf;
  int err=read(fd1,&buf,sizeof(char));
}
int main(…) {

 fd1=open("t1.dat",O_RDWR);
 int err=pthread_create(&(t1),NULL,&f1,NULL);
```

wooclap https://app.wooclap.com/QEAHMX

▶ **Que se passe t-il dans le thread principal et dans le thread créé**

Passer plusieurs réels à un thread

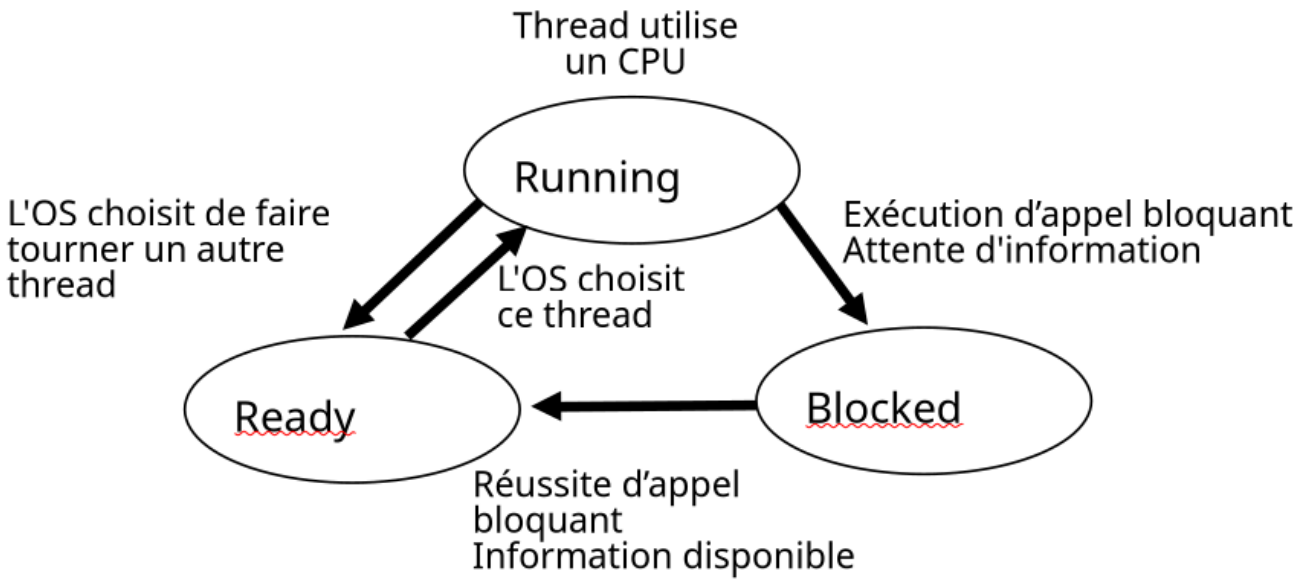
```c
    pthread_t pt;
    double tab[2]={2.0,3.0}; //Utiliser un tableau

    void *f3(void *param) {
    printf("f3\n");
    double *p=(double *) param;
    printf("%f %f\n",*p,*p+1);
    return NULL;
    }
    int main( int argc, char **argv) {
    int err=pthread_create(&pt,NULL,&f3,(void *)tab)
```

```c
pthread_t pt;
struct pair { // Utiliser une structure
  double x;
  double y;
};
void *f2(void *param) {
  struct pair *p=(struct pair *) param;
  printf("%f %f\n",p->x,p->y);
  return NULL;
}
int main( int argc, char **argv) {
struct pair * a =(struct pair *) malloc(sizeof(struct pair));
a->x=2.3; a->y=4.5;
int err=pthread_create(&pt,NULL,&f2,(void *)a);
```

Attention à printf

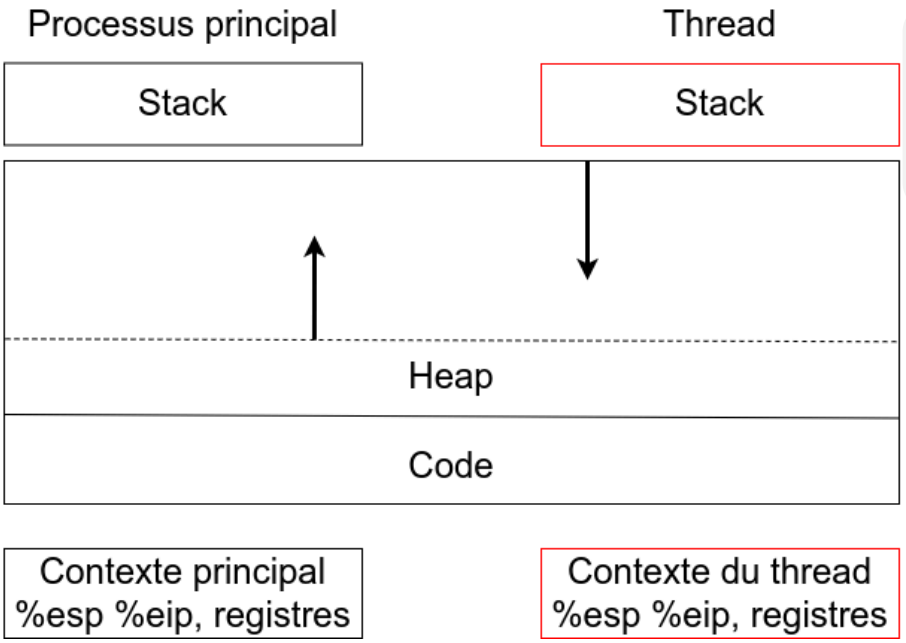
```
Méfiez-vous !
    - Tous les threads partagent stdout, mais printf utilise un buffer qui n'envoie
    - des données sur stdout que lorsqu'il y en a suffisamment
    - man fflush
```

Les états d'un thread



Mutex et sémaphores

Communication entre threads




```
En utilisant les zones mémoires accessibles aux deux threads
    - variables globales
    - heap
```

Les dangers avec les threads

Violation d'exclusion mutuelle

- Deux threads modifient la même zone mémoire sans coordination

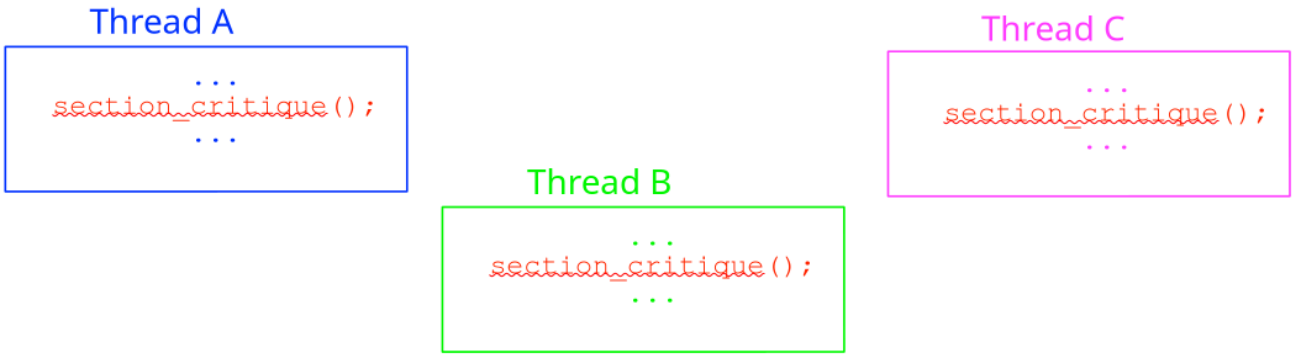
```c
long global=0;
int increment(int i) {
  return i+1;
}
void *func(void * param) {
  for(int j=0;j<1000000;j++) {
    global=increment(global);
 }
  return(NULL);
}
```

```
for i in {1..5}; do ./pthread-test;  done
    global: 3408577
    global: 3175353
    global: 1994419
    global: 3051040
    global: 2118713
```

Deadlock

L'ensemble du programme est bloqué en attente sur des mutex ou des sémaphore

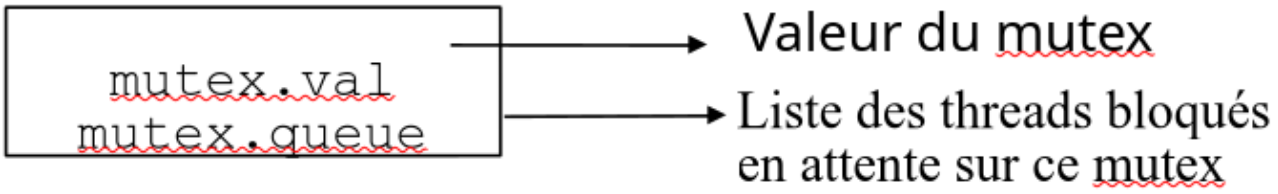
Problème de l'exclusion mutuelle: les sections critiques




```
Conditions à remplir par une solution
    - Deux threads ne peuvent y être en même temps
    - Un thread se trouvant hors de sa section critique ne peut pas bloquer un autr
e thread
    - Un thread ne doit pas attendre indéfiniment le droit d'entrer dans sa section
critique
    - Aucune hypothèse n'est faite sur la vitesse des threads ou le nombre de CPUs
```

Mutex posix

```
Structure de données spéciale de la librairie threads POSIX associée à une ressou
rce
    - libre (unlocked en anglais)
    - réservée (locked en anglais)
```



Les opérations sur les mutex

```
3 opérations possibles sur un mutex
    1. Initialisation à unlocked
    2. lock(m)
    3. unlock(m)
```

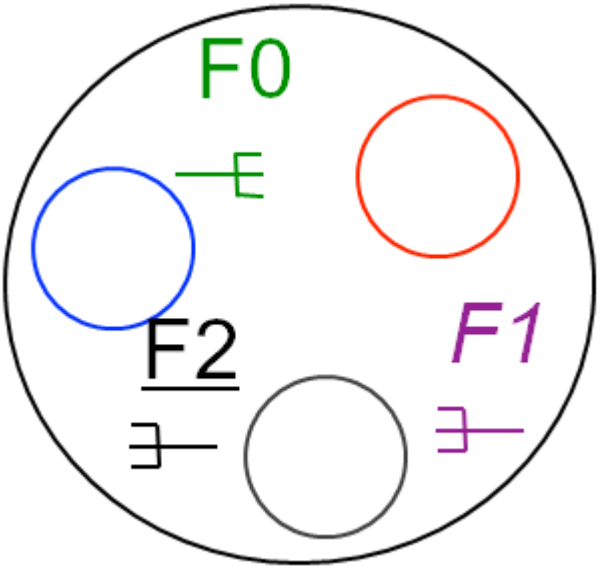
```
pthread_mutex_lock(…)
pthread_mutex_unlock(…)
pthread_mutex_init(…)
```

wooclap https://app.wooclap.com/QEAHMX

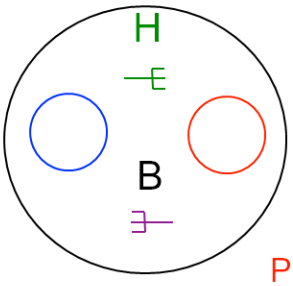
▶ **Lesquelles de ces fonctions pourraient bloquer le thread qui les exécute ?**

Le diner des philosophes

```
N philosophes doivent se partager un repas dans une salle de méditation
    - La table contient N fourchettes et N assiettes
    - Chaque philosophe a une place réservée et a besoin pour manger de
        - La fourchette à sa gauche
        - La fourchette à sa droite
```



Problème avec deux philosophes



Philosophe1

```
while(TRUE)
{
 think();
 lock(H);  /* fourchette H */
 lock(B);  /* fourchette B */
 eat();
 unlock(B); /* fourchette B */
 unlock(H); /* fourchette H */
}
```

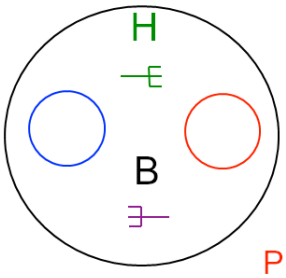
Philosophe2

```
while(TRUE)
{
 think();
 lock(B);  /* fourchette B */
 lock(H);  /* fourchette H */
 eat();
 unlock(H); /* fourchette H */
 unlock(B); /* fourchette B */
}
```

wooclap https://app.wooclap.com/QEAHMX

▶ **Y a-t-il un deadlock dans le code de ces 2 philosophes ?**

Problème avec deux philosophes sans deadlock



Philosophe1

```
while(TRUE)
{
 think();
 lock(B);  /* fourchette B */
 lock(H);  /* fourchette H */
 eat();
 unlock(B); /* fourchette B */
 unlock(H); /* fourchette H */
}
```

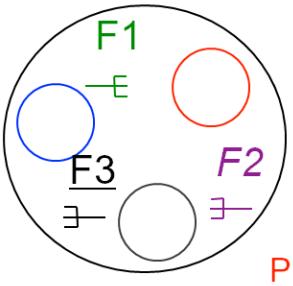
Philosophe2

```
while(TRUE)
{
 think();
 lock(B);  /* fourchette B */
 lock(H);  /* fourchette H */
 eat();
 unlock(H); /* fourchette H */
 unlock(B); /* fourchette B */
}
```

Problème avec trois philosophes

Philosophe1

```
while(TRUE)
{
 think();
 lock(F1);  /* left fork */
 lock(F3);  /* right fork */
 eat();
 unlock(F1); /* left fork */
 unlock(F3); /* right fork */
}
```



Philosophe2

```
while(TRUE)
{
 think();
 lock(F2);  /* left fork */
 lock(F1);  /* right fork */
 eat();
 unlock(F2); /* left fork */
 unlock(F1); /* right fork */
}
```

Philosophe3

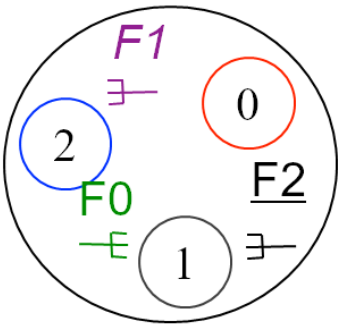
```
while(TRUE)
{
 think();
 lock(F3);  /* left fork */
 lock(F2);  /* right fork */
 eat();
 unlock(F3); /* left fork */
 unlock(F2); /* right fork */
}
```

wooclap https://app.wooclap.com/QEAHMX

▶ **Y a-t-il un deadlock dans le code de ces 3 philosophes ?**

Problème avec N philosophes sans deadlock

```
#define LEFT (i+N-1)%N
#define RIGHT (i+1)%N
/* Initialisation */
mutex fork[N]=1;
```



Philosophe[i]

```
while(TRUE)
{
 think();
 /* take forks */
 take_forks(LEFT,RIGHT);
 /* forks taken */
 eat();
 /* release forks */
 unlock(&fork[LEFT]);
 unlock(&fork[RIGHT]);
 /* forks released */
}
```

```
void take_forks(int i, int j)
{
 if(i<j)
 {
   lock(&fork[i]);
   lock(&fork[j]);
 }
 else
 {
   lock(&fork[j]);
   lock(&fork[i]);
 }
}
```

Les sémaphores

```
3 opérations possibles sur un mutex
    1. Init(s) A une valeur entière non-négative
    2. Down(s) parfois appelé wait(s) ou P(s)
    3. Up(s) parfois appelé signal(s)/post(s) ou V(s)
```

```
down(semaphore s)
{
 s.val=s.val-1;
 if(s.val<0)
 {
   Place this thread in s.queue;
   This thread is blocked;
 }
}
```

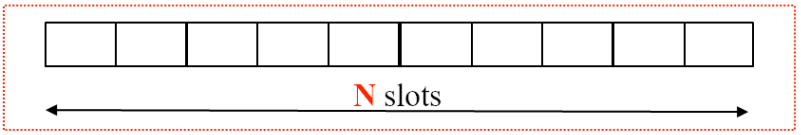
```
up(semaphore s)
{
 s.val=s.val+1;
 if(s.val<=0)
 {
   Remove one thread(T) from s.queue;
   Mark Thread(T) as ready to run;
 }
}
```

Appel system pour les semaphores

```
    int sem_init(sem_t *sem, int pshared, unsigned int value); // Initialise le n
ombre de lock
    int sem_wait(sem_t *sem);  // Decrémente le nombre de lock
    int sem_trywait(sem_t *sem); // Decrémente le nombre de lock
    int sem_post(sem_t *sem); // Incrémente le nombre de lock
    }
```

Le problème du producteur-consommateur

```
/* Initialisation */
pthread_lock_t mutex;
sem_t empty; // init N
sem_t full; // init 0
```



Producteur

```
Void producer(void)
{
 int item;
 while(true)
 {
  item=produce(item);
  sem_wait(&empty);
  pthread_mutex_lock(&mutex);
  insert_item();
  pthread_mutex_unlock(&mutex);
  sem_post(&full);
 }
}
```

Consommateur

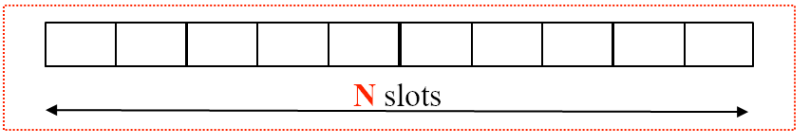
```
void consumer(void)
{
 int item;
 while(true)
 {
  sem_wait(&full);
  pthread_mutex_lock(&mutex);
  item=remove(item);
  pthread_mutex_unlock(&mutex);
  sem_post(&empty);
 }
```

wooclap https://app.wooclap.com/QEAHMX

▶ **Y a-t-il un deadlock dans le code du producteur-consommateurs ?**

Le problème du producteur-consommateur sans deadlock

```
/* Initialisation */
pthread_lock_t mutex;
sem_t empty; // init N
sem_t full; // init 0
```



Producteur

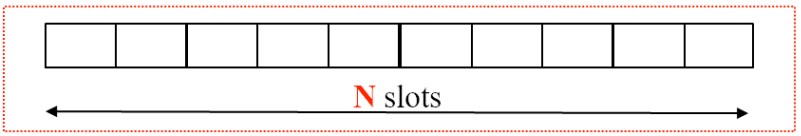
```
Void producer(void)
{
 int item;
 while(true)
 {
  item=produce(item);
  pthread_mutex_lock(&mutex);
  sem_wait(&empty);
  insert_item();
  pthread_mutex_unlock(&mutex);
  sem_post(&full);
 }
}
```

Consommateur

```
void consumer(void)
{
 int item;
 while(true)
 {
  pthread_mutex_lock(&mutex);
  sem_wait(&full);
  item=remove(item);
  pthread_mutex_unlock(&mutex);
  sem_post(&empty);
 }
}
```

Le problème du producteur-consommateur

```
/* Initialisation */
pthread_lock_t mutex;
sem_t empty; // init N
sem_t full; // init 0
```



Producteur

```
Void producer(void)
{
 int item;
 while(true)
 {
  item=produce(item);
  sem_wait(&empty);
  pthread_mutex_lock(&mutex);
  insert_item();
  sem_post(&full);
  pthread_mutex_unlock(&mutex);
 }
}
```

Consommateur

```
void consumer(void)
{
 int item;
 while(true)
 {
  sem_wait(&full);
  pthread_mutex_lock(&mutex);
  item=remove(item);
  sem_post(&empty);
  pthread_mutex_unlock(&mutex);
 }
}
```

wooclap https://app.wooclap.com/QEAHMX

▶ **Y a-t-il un deadlock dans le code du producteur-consommateurs ?**

Seconde phase du projet

Réduire le temps de calcul

```
1. Activer les optimisations du compilateur
    - Multiplication de matrices (1000x1000):
    - gcc mat.c -> exécution en 6.3 secondes
    - gcc -Ofast mat.c -> exécution en 1.6 secondes
2. Utiliser le plus efficacement les quatre cœurs
    - Réduire les lock/wait au strict nécessaire, sans risquer de violation de sect
ion critique
    - Vos threads doivent être les plus indépendants possibles pour être efficaces
    - Vous avez quatre cœurs, ce n'est probablement pas utile de lancer 32 threads,
mais peut-être que 6 ou 8 pourraient vous donner de bons résultats
3. Utiliser le plus efficacement la mémoire cache
    - Un code est plus efficace si il utilise travaille sur des zones de mémoire pr
oches
    - Les éléments d'une structure sont stockés à des adresses proches en mémoire
```

Consommation du raspberry pi 3B+

source

Pi Model	Desktop Idle (no WiFi)	1080p Video (no WiFi)	Desktop Idle (WiFi)	1080p Video (WiFi)	YouTube Video (WiFi)
Pi Zero W	220mA	245mA	250mA	270mA	-
A+	225mA	245mA	-	-	-
3 A+	305mA	320mA	340mA	355mA	400mA
B+	320mA	340mA	-	-	-
2 B	345mA	370mA	-	-	-
3 B	360mA	390mA	390mA	420mA	450mA
3 B+	510mA	520mA	550mA	585mA	600mA

source

Raspberry Pi 3 B+

Pi State	Power Consumption
Idle	350 mA (1.9 W)
`ab -n 100 -c 10` (uncached)	950 mA (5.0 W)
400% CPU load (`stress --cpu 4`)	980 mA (5.1 W)

Optimisations possibles

```
1. Désactiver ce qui est inutile
    - HDMI
    - Réseau ?
    - USB  ?
    - Gardez l'accès à la machine!
2. Ajuster la fréquence du CPU
    - Un CPU moins rapide peut consommer moins d'énergie qu'un CPU rapide
3.Désactiver un ou plusieurs coeurs
```

source

source

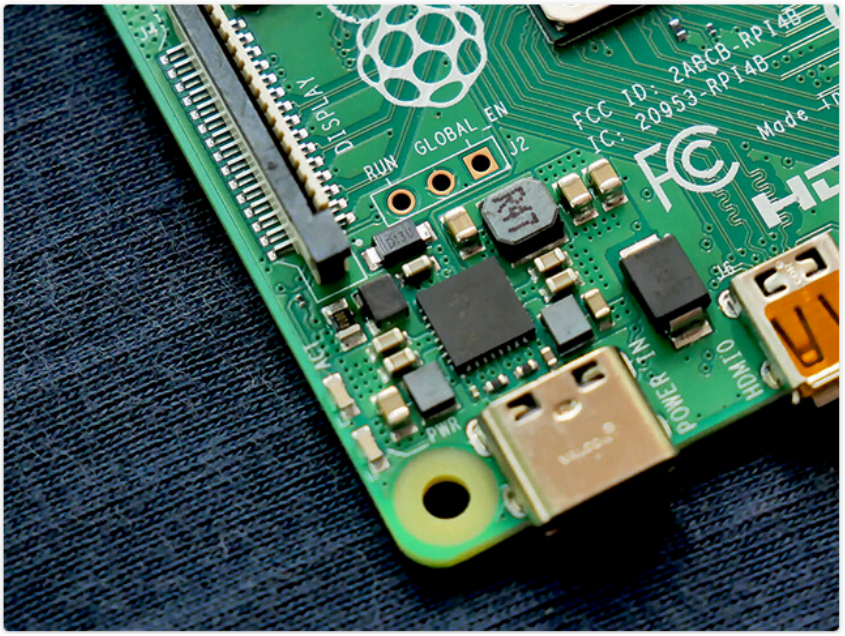
Blog > Optimizing Raspberry Pi Power Consumption

Optimizing Raspberry Pi Power Consumption

> *Tips and tricks for optimizing the power consumption of a Raspberry Pi to make it a candidate for off-grid remote monitoring.*

Rob Lauer • April 19, 2023

Raspberry Pi Low Power



Réduire la consommation de mémoire

```
1. Compacter les structures de données utilisées
2. Stocker intelligemment les matrices creuses
    - Si une matrice a 90% de zéros, inutile de stocker tous ces zéros en mémoire
    - Vous devez avoir le même résultat numérique !
3. Réduire la taille du programme
    - Différentes stratégies possibles, optimisation du compilateur, retirer une pa
rtie des librairies
    - https://interrupt.memfault.com/blog/code-size-optimization-gcc-flags
4. Allouer la mémoire par blocs plutôt que globalement au début du programme
```

Table of Contents