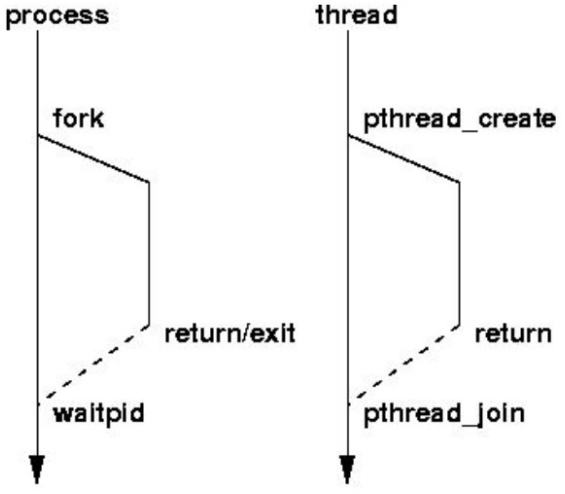
Funcion de creación de un hilo

```c
#include <pthread.h>

int pthread_create(
                   pthread_t * thread,                  /* out */
                   const pthread_attr_t * attr,         /* in */
                   void *(* start )(void *),            /* in */
                   void * arg                           /* in */
                  );
```




```c
void* print_xs (void* unused)
{
  while (1)
    fputc ('x', stderr);
  return NULL;
}
```

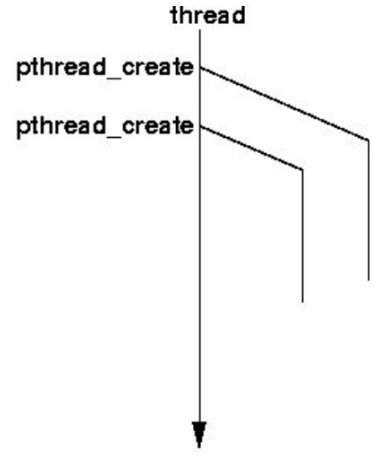
```c
int main ()
{
  pthread_t thread_id;
  /* Create a new thread.  The new thread will run the print_xs
     function.  */
  pthread_create (&thread_id, NULL, &print_xs, NULL);
```

Funcion de creación de un hilo

```c
#include <pthread.h>
#include <stdio.h>

/* Prints x's to stderr.  The parameter is unused.  Does not return.  */

void* print_xs (void* unused)
{
  while (1)
    fputc ('x', stderr);
  return NULL;
}

/* The main program.  */

int main ()
{
  pthread_t thread_id;
  /* Create a new thread.  The new thread will run the print_xs
     function.  */
  pthread_create (&thread_id, NULL, &print_xs, NULL);
  /* Print o's continuously to stderr.  */
  while (1)
    fputc ('o', stderr);
  return 0;
}
```

```c
#include <pthread.h>

int pthread_create(
                   pthread_t * thread,                  /* out */
                   const pthread_attr_t * attr,         /* in */
                   void *(* start )(void *),            /* in */
                   void * arg                           /* in */
                  );
```

```c
#include <pthread.h>

int pthread_create(
                   pthread_t * thread,                  /* out */
                   const pthread_attr_t * attr,         /* in */
                   void *(* start )(void *),            /* in */
                   void * arg                           /* in */
                  );
```




```c
struct char_print_parms
{
  /* The character to print.  */
  char character;
  /* The number of times to print it.  */
  int count;
};

void* char_print (void* parameters)
{
  /* Cast the cookie pointer to the right type.  */
  struct char_print_parms* p = (struct char_print_parms*) parameters;
  int i;

  for (i = 0; i < p->count; ++i)
    fputc (p->character, stderr);
  return NULL;
}
```

```c
int main ()
{
  pthread_t thread1_id;
  pthread_t thread2_id;
  struct char_print_parms thread1_args;
  struct char_print_parms thread2_args;

  /* Create a new thread to print 30000 x's.  */
  thread1_args.character = 'x';
  thread1_args.count = 30000;
  pthread_create (&thread1_id, NULL, &char_print, &thread1_args);

  /* Create a new thread to print 20000 o's.  */
  thread2_args.character = 'o';
  thread2_args.count = 20000;
  pthread_create (&thread2_id, NULL, &char_print, &thread2_args);

  /*-------INSERTAR AQUÍ-------*/

  return 0;
}
```

```c
struct char_print_parms
{
  /* The character to print.  */
  char character;
  /* The number of times to print it.  */
  int count;
};
```

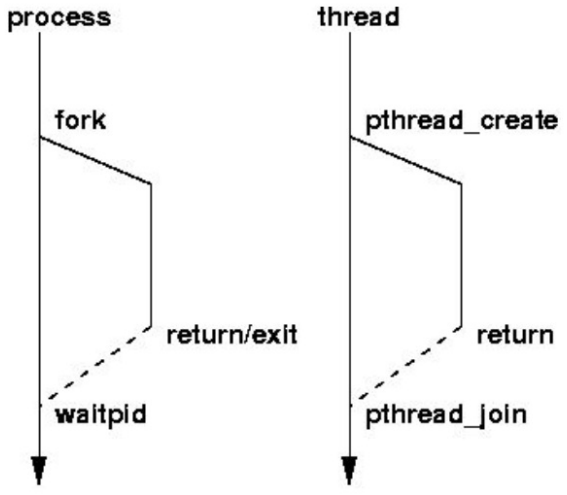
```c
#include <pthread.h>

int pthread_create(
                   pthread_t * thread,                /* out */
                   const pthread_attr_t * attr,       /* in */
                   void *(* start )(void *),          /* in */
                   void * arg                         /* in */
                  );
```

```c
void* char_print (void* parameters)
{
  /* Cast the cookie pointer to the right type.  */
  struct char_print_parms* p = (struct char_print_parms*) parameters;
  int i;

  for (i = 0; i < p->count; ++i)
    fputc (p->character, stderr);
  return NULL;
}
```

```c
int main ()
{
  pthread_t thread1_id;
  pthread_t thread2_id;
  struct char_print_parms thread1_args;
  struct char_print_parms thread2_args;

  /* Create a new thread to print 30000 x's.  */
  thread1_args.character = 'x';
  thread1_args.count = 30000;
  pthread_create (&thread1_id, NULL, &char_print, &thread1_args);

  /* Create a new thread to print 20000 o's.  */
  thread2_args.character = 'o';
  thread2_args.count = 20000;
  pthread_create (&thread2_id, NULL, &char_print, &thread2_args);

  /*-------INSERTAR AQUÍ-------*/

  return 0;
}
```

```
(base) tigarto@fuck-pc:~/Documents/UdeA/sistemas-o
s_viejos/lab8/ejemplos_hilos/parte1$ ./a.out
xx(base) tigarto@fuck-pc:~/Documents/UdeA/sistemas
s_viejos/lab8/ejemplos_hilos/parte1$ ./a.out
(base) tigarto@fuck-pc:~/Documents/UdeA/sistemas-o
s_viejos/lab8/ejemplos_hilos/parte1$ ./a.out
(base) tigarto@fuck-pc:~/Documents/UdeA/sistemas-o
s_viejos/lab8/ejemplos_hilos/parte1$ ./a.out
(base) tigarto@fuck-pc:~/Documents/UdeA/sistemas-o
s_viejos/lab8/ejemplos_hilos/parte1$
```

ptreahd_join




```c
#include <pthread.h>

int pthread_create(
                   pthread_t * thread,                  /* out */
                   const pthread_attr_t * attr,         /* in */
                   void *(* start )(void *),            /* in */
                   void * arg                           /* in */
                  );
```

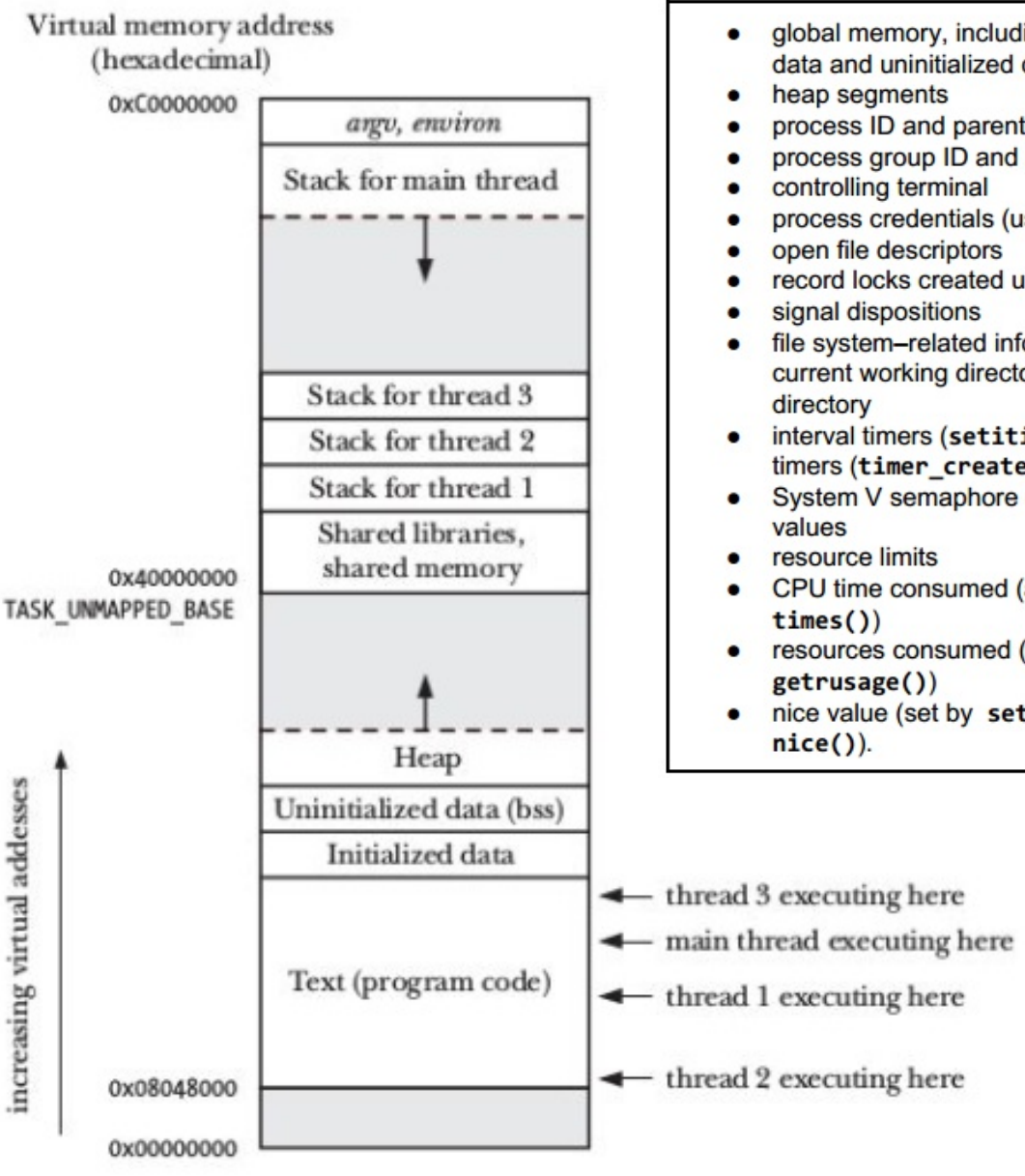
```c
include <pthread.h>

int pthread_join(
                  pthread_t thread ,              /* in */
                  void ** retval                  /* out */
                 );
```

```c
int main ()
{
  pthread_t thread1_id;
  pthread_t thread2_id;
  struct char_print_parms thread1_args;
  struct char_print_parms thread2_args;

  /* Create a new thread to print 30000 x's.  */
  thread1_args.character = 'x';
  thread1_args.count = 30000;
  pthread_create (&thread1_id, NULL, &char_print, &thread1_args);

  /* Create a new thread to print 20000 o's.  */
  thread2_args.character = 'o';
  thread2_args.count = 20000;
  pthread_create (&thread2_id, NULL, &char_print, &thread2_args);

  /*-------  /*-------INSERTAR AQUÍ-------*/

  return 0;
}          pthread_join (thread1_id, NULL);
           pthread_join (thread2_id, NULL);
```

PID e hilos

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

void* thread_function (void* arg)
{
  fprintf (stderr, "child thread pid is %d\n", (int) getpid ());
  /* Spin forever.  */
  while (1);
  return NULL;
}

int main ()
{
  pthread_t thread;
  fprintf (stderr, "main thread pid is %d\n", (int) getpid ());
  pthread_create (&thread, NULL, &thread_function, NULL);
  /* Spin forever.  */
  while (1);
  return 0;
}
```

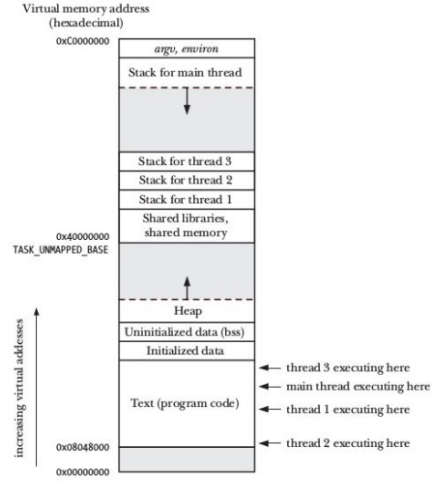
```
s_viejos/lab8/ejemplos_hilos/parte1$ gcc lab4_p1_example4.c -lpthread
(base) tigarto@fuck-pc:~/Documents/UdeA/sistemas-operativos/presencial
s_viejos/lab8/ejemplos_hilos/parte1$ ./a.out
main thread pid is 26262
child thread pid is 26262
^C
```



Parte 1 - Compartido	Parte 2 - Individual
• global memory, including the initialized data and uninitialized data • heap segments • process ID and parent process ID • process group ID and session ID • controlling terminal • process credentials (user and group IDs) • open file descriptors • record locks created using **`fcntl()`** • signal dispositions • file system–related information: umask, current working directory, and root directory • interval timers (**`setitimer()`**) and POSIX timers (**`timer_create()`**) • System V semaphore undo (**`semadj`**) values • resource limits • CPU time consumed (as returned by **`times()`**) • resources consumed (as returned by **`getrusage()`**) • nice value (set by **`setpriority()`** and **`nice()`**).	• thread ID • signal mask • thread-specific data • alternate signal stack (**`signaltstack()`**) • the errno variable • floating-point environment • realtime scheduling policy and priority • CPU affinity (Linux-specific) • capabilities (Linux-specific) • stack (local variables and function call linkage information).

```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

/* Global variable */
int x;

void fd(void);

int main(void) {
  pthread_t threads_ids[4];
  int i;
  for(i = 0; i < 4; i++) {
    pthread_create(&threads_ids[i],NULL,(void *)fd, NULL);
    printf("Iniciando hilo: %d\n", i + 1);
  }
  for(i = 0; i < 4; i++) {
    pthread_join(threads_ids[i],NULL);
  }
}
void fd(void) {
  int i;
  printf("Thread PID: %lu \n-> x = %d (before to be incremented 1000 times for this
thread)\n", (unsigned long)pthread_self(),x);
  for(i = 1;i <=1000;i++) {
    x++;
  }
}
```

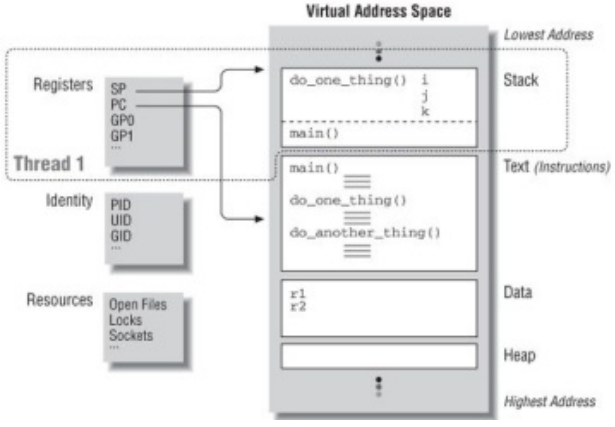
```
s_viejos/lab8/ejemplos_hilos/parte1$ ./a.out
Iniciando hilo: 1
Thread PID: 140071992370944
-> x = 0 (before to be incremented 1000 times for this thread)
Iniciando hilo: 2
Thread PID: 140071983978240
-> x = 1000 (before to be incremented 1000 times for this thread)
Iniciando hilo: 3
Thread PID: 140071975585536
-> x = 2000 (before to be incremented 1000 times for this thread)
Iniciando hilo: 4
Thread PID: 140071967192832
-> x = 3000 (before to be incremented 1000 times for this thread)
(base) tigarto@fuck-pc:~/Documents/UdeA/sistemas-operativos/presencial/2020/repo
s_viejos/lab8/ejemplos_hilos/parte1$ ./a.out
Iniciando hilo: 1
Thread PID: 140106661558016
-> x = 0 (before to be incremented 1000 times for this thread)
Iniciando hilo: 2
Thread PID: 140106653165312
-> x = 1000 (before to be incremented 1000 times for this thread)
Iniciando hilo: 3
Iniciando hilo: 4
Thread PID: 140106636379904
-> x = 2000 (before to be incremented 1000 times for this thread)
Thread PID: 140106644772608
-> x = 2095 (before to be incremented 1000 times for this thread)
(base) tigarto@fuck-pc:~/Documents/UdeA/sistemas-operativos/presencial/2020/repo
```



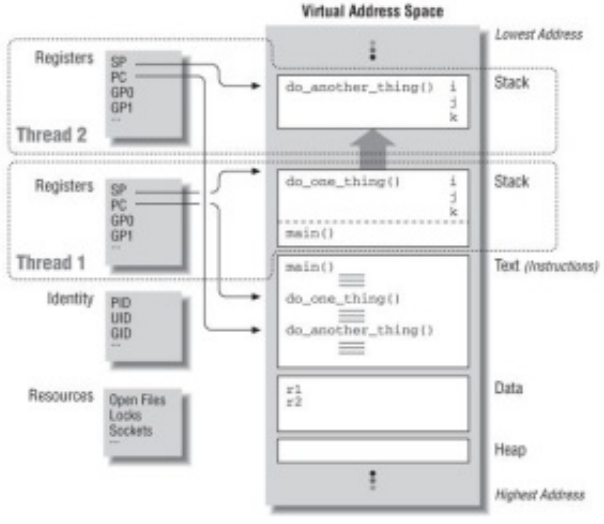
Retornando un valor desde el hilo

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
/* Funcion para calcular primo.*/
void* calcular_primo (void* arg);

int main(){
  pthread_t id_hilo;
  int cual_primo = 5000;
  int* primo;
  // Inicia el hilo, se requiere el 5000-iesimo numero primo
  pthread_create(&id_hilo, NULL, &calcular_primo, &cual_primo);
  // Puedo hacer algo mientras... si quiero
  // Espero que el número sea calculado y retornado
  pthread_join(id_hilo, (void *) &primo);
  // Imprimo el número entregado
  printf("El %d-esimo número primo es %d\n", cual_primo, *primo);
  free(primo);
  return 0;
}

/* Calcula los numeros primos sucesivamente. retorna el n-esimo numero primo donde n es el
valor apuntado por arg*/
void* calcular_primo (void* arg){
  int candidato = 2;
  int n = *((int*)arg);
  int factor;
  int es_primo;
  while(1){
    es_primo = 1;
    for(factor = 2; factor < candidato; factor++)
      if(candidato % factor == 0){
       es_primo = 0;
       break;
      }
    if(es_primo){
      if (--n == 0){
       int* p_c = malloc(sizeof(int));
       *p_c = candidato;
       return p_c;
      }
    }
    candidato++;
  }
  return NULL;
}
```

```
s_viejos/lab8/ejemplos_hilos/parte1$ ./a.out
El 5000-esimo número primo es 48611
```




```
main(void)
{
  do_one_thing(&r1);
  do_another_thing(&r2);
  do_wrap_up(r1, r2);
  return 0;
}

void do_one_thing(int *pnum_times)
{
  int i, j, x;

  for (i = 0;    i  <  4;  i++) {
    printf("doing one thing\n");
    for (j = 0; j < 10000; j++) x = x + i;
    (*pnum_times)++;
  }
}

void do_another_thing(int *pnum_times)
{
  int i, j, x;

  for (i = 0;    i  <  4;  i++) {
    printf("doing another \n");
    for (j = 0; j < 10000; j++) x = x + i;
    (*pnum_times)++;
  }
}
```




```
void do_one_thing(int *);
void do_another_thing(int *);
void do_wrap_up(int, int);

int r1 = 0, r2 = 0;

extern int main(void)
{
  pthread_t       thread1,  thread2;

  pthread_create(&thread1,
          NULL,
          (void *) do_one_thing,
          (void *) &r1);

  pthread_create(&thread2,
          NULL,
          (void *) do_another_thing,
          (void *) &r2);

  pthread_join(thread1,  NULL);
  pthread_join(thread2,  NULL);

  do_wrap_up(r1, r2);
  return 0;
}

void do_one_thing(int *pnum_times)
{
  int i, j, x;

  for (i = 0;    i  <  4;  i++) {
    printf("doing one thing\n");
    for (j = 0; j < 10000; j++) x = x + i;
    (*pnum_times)++;
  }
}
```

Diferenciando procesos respecto a hilos