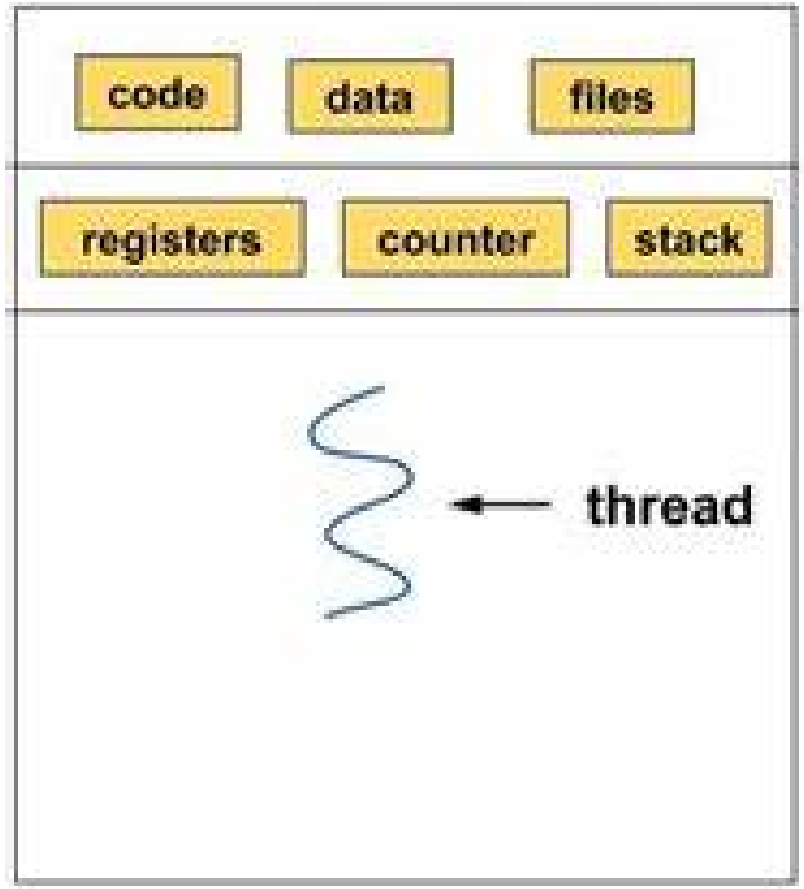
Threads

- A thread is an execution unit that has its own program counter, a stack and a set of registers that reside in a process .
- Threads can’t exist outside any process.
- Each thread belongs to exactly one process.
- The information like code segment, files, and data segment can be shared by the different threads.

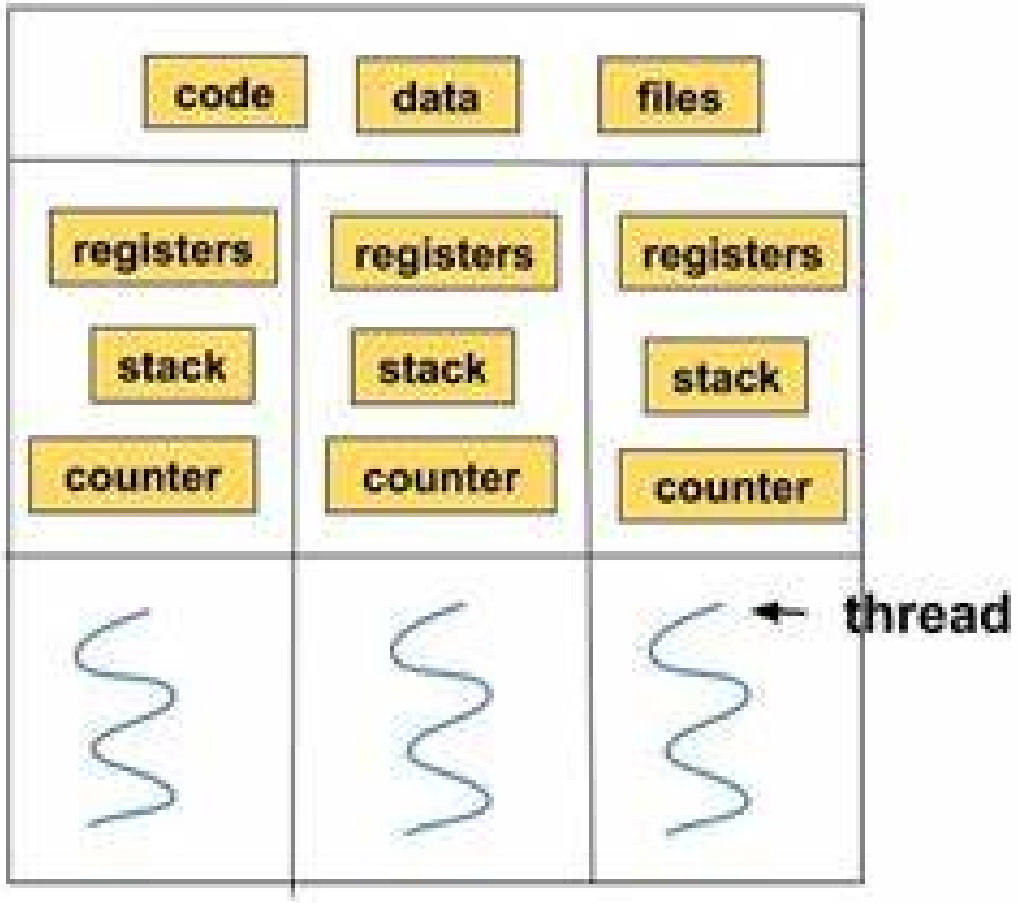
Threads are popularly used to improve the application through **parallelism** .

Actually only one thread is executed at a time by the CPU, but the **CPU switches rapidly** between the threads to give an illusion that the threads are running parallelly.

Threads are also known as light-weight processes.



Single-threaded process



Multi-threaded process

- Pthreads refers to the POSIX standard (IEEE 1003.1c) defining an API for thread creation and synchronization.

- This defines specification for thread behaviour, not an implementation.

- The specification can be implemented by OS designers in any way they wish. So many systems implement the Pthreads specification; most are UNIX-type systems, including Linux, Mac OS X, and Solaris.

POSIX Thread Functions

Name	Description
pthread_attr_init	Initialize a thread attribute object
pthread_cancel	Terminate another thread
pthread_create	Create a thread
pthread_detach	Set thread to release resources
pthread_equal	Test two thread IDs for equality
pthread_exit	Exit a thread without exiting the process
pthread_kill	Send a signal to a thread
pthread_join	Wait for a thread to terminate
pthread_self	Find out own thread ID

pthread_create - thread creation

#include <pthread.h>

int pthread_create(pthread_t *_thread_, const pthread_attr_t *_attr_,
void *(*_start_routine_)(void*), void *_arg_);

- The *pthread_create()* function is used to create a new thread, with attributes specified by *attr*, within a process.
- If *attr* is NULL, the default attributes are used.

- Upon successful completion, *pthread_create()* stores the ID of the created thread in the location referenced by *thread*.
- If successful, the *pthread_create()* function returns zero. Otherwise, an error number is returned to indicate the error.

The thread is created executing *start_routine* with *arg* as its sole argument.

If the *start_routine* returns, the effect is as if there was an implicit call to *pthread_exit()* using the return value of *start_routine* as the exit status..

On success, **pthread_create**() returns 0; on error, it returns an error number, and the contents of **thread* are undefined

int pthread_join(pthread_t *thread*, void ***retval*);

The **pthread_join**() function waits for the thread specified by *thread* to terminate.

If that thread has already terminated, then **pthread_join**() returns immediately.

The thread specified by *thread* must be joinable.

void pthread_exit(void *_retval_);

The **pthread_exit**() function terminates the calling thread and returns a value via *retval* that (if the thread is joinable) is available to another thread in the same process that calls **pthread_join**(3).

```
#include <stdio.h>

#include <pthread.h>

// Function executed by each thread

void *printGoodMorning(void *threadId)

 {    long tid = (long)threadId;

      printf("Good morning from thread %ld\n", tid);

      pthread_exit(NULL);
 }
```

```
int main()
{    pthread_t threads[4];
    int i;
    for (i = 0; i < 4; i++)
        {       int rc = pthread_create(&threads[i], NULL, printGoodMorning,
(void *) i);
                if (rc)
                   {     printf("Error creating thread %d\n", i);
                         return -1;
                   }
        }
```

```
// Wait for all threads to finish

  for (i = 0; i < 4; i++)

    {       pthread_join(threads[i], NULL);

    }

 return 0;
}
```

Good morning from thread 0

Good morning from thread 1

Good morning from thread 2

Good morning from thread 3

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#define MAT_SIZE 10
#define MAX_THREADS 100
```

```
int i,j,k,x,y,z,a;//Parameters For Rows And Columns

int matrix1[MAT_SIZE][MAT_SIZE]; //First Matrix

int matrix2[MAT_SIZE][MAT_SIZE]; //Second Matrix

int result [MAT_SIZE][MAT_SIZE]; //Multiplied Matrix

void* mul(void*);
```

```
typedef struct parameters
    {     int x,y;
    }    args;

void* mul(void*arg)
{   args* p=arg;
  for(int a=0;a<j;a++)
       {     result[p->x][p->y]+=matrix1[p->x][a]*matrix2[a][p->y];
       }
      sleep(3);
   pthread_exit(0);
}
```

```
int main()
{   for(int x=0;x<10;x++)
      {      for(int y=0;y<10;y++)
           {         // matrix1[x][y] = 0;
                     //  matrix2[x][y] = 0;
                          result[x][y] = 0;
           }
      }
```

```
printf("Enter number of rows for matrix 1: ");
 scanf("%d",&i);
 printf("Enter number of columns for matrix 1: ");
 scanf("%d",&j);   printf("\n --- reading  Matrix 1 ---\n\n");
 for(int x=0;x<i;x++)
    {     for(int y=0;y<j;y++)
             {       printf("Enter variable [%d,%d]: ",x+1,y+1);
                     scanf("%d",&matrix1[x][y]);
             }
    }
```

Read another matrix

Display both input matrices

```
pthread_t thread[MAX_THREADS];

int thread_number = 0;

args p[i*k];
```

```
for(int x=0;x<i;x++)
   {     for(int y=0;y<k;y++)
         {   p[thread_number].x=x;
             p[thread_number].y=y;
             int status;
             status = pthread_create(&thread[thread_number], NULL, mul, (void *)
                                                                    &p[thread_number]);

             if(status!=0)
                  {    printf("Error In Threads");
                       exit(0);
                  }
            thread_number++;
        }
```

```
for(int z=0;z<(i*k);z++)

    {
        pthread_join(thread[z],NULL );
    }
```

```
printf(" --- Multiplied Matrix ---\n\n");
for(int x=0;x<i;x++)
        {     for(int y=0;y<k;y++)
                  printf("%5d",result[x][y]);

                  printf("\n\n");
        }
```

```
printf(" ---> Used Threads : %d \n\n",thread_number);

for(int z=0;z<thread_number;z++)

    printf(" - Thread %d ID : %d\n",z+1,(int)thread[z]);

return 0;

}
```