FORK VS THREAD

Thread Creation

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <pthread.h>    //Thread header file
int x = 2;              // Threads will share this data
void *runner1();        // Thread1 call this function
void *runner2();        // Thread2 call this function

int main () {
pthread_t tid1, tid2;       //Thread ids
pthread_attr_t attr;        // Set of thread attributes
pthread_attr_init(&attr); //Set the default attr of the thread
pthread_create(&tid1, &attr, runner1, NULL);
pthread_create(&tid2, &attr, runner2, NULL);
pthread_join(tid1, NULL);   //wait for the thread to exit
pthread_join(tid2, NULL);   //wait for the thread to exit
 ++x;
printf("Value of x by main %d, Process id of main %d\n",  x, getpid());
return 0;
}
```

```
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
#include<sys/types.h>
int x =2;

int main(){
   fork();
   fork();
   ++x;
   sleep(2);
   printf("Value of x: %d and process id:
                              %d\n", x,getpid());
   return 0;
   }
```

Child Creation

Thread Creation

```
void *runner1() // Thread 1 will execute in this function
{
    ++x;

    printf("Value of x by thread1 %d, Process id of thread1 %d\n",  x, getpid());
    sleep(2);
    pthread_exit(0);
}

void *runner2()    // Thread 2 will execute in this function
{
    ++x;

    printf("Value of x by thread2 %d, Process id of thread2 %d\n",  x, getpid());
    sleep(2);
    pthread_exit(0);
}
```

Thread Output

```
Value of x by thread1 3, Process id of thread1 1645
Value of x by thread2 4, Process id of thread2 1645
Value of x by main 5, Process id of main 1645
```

Fork Output

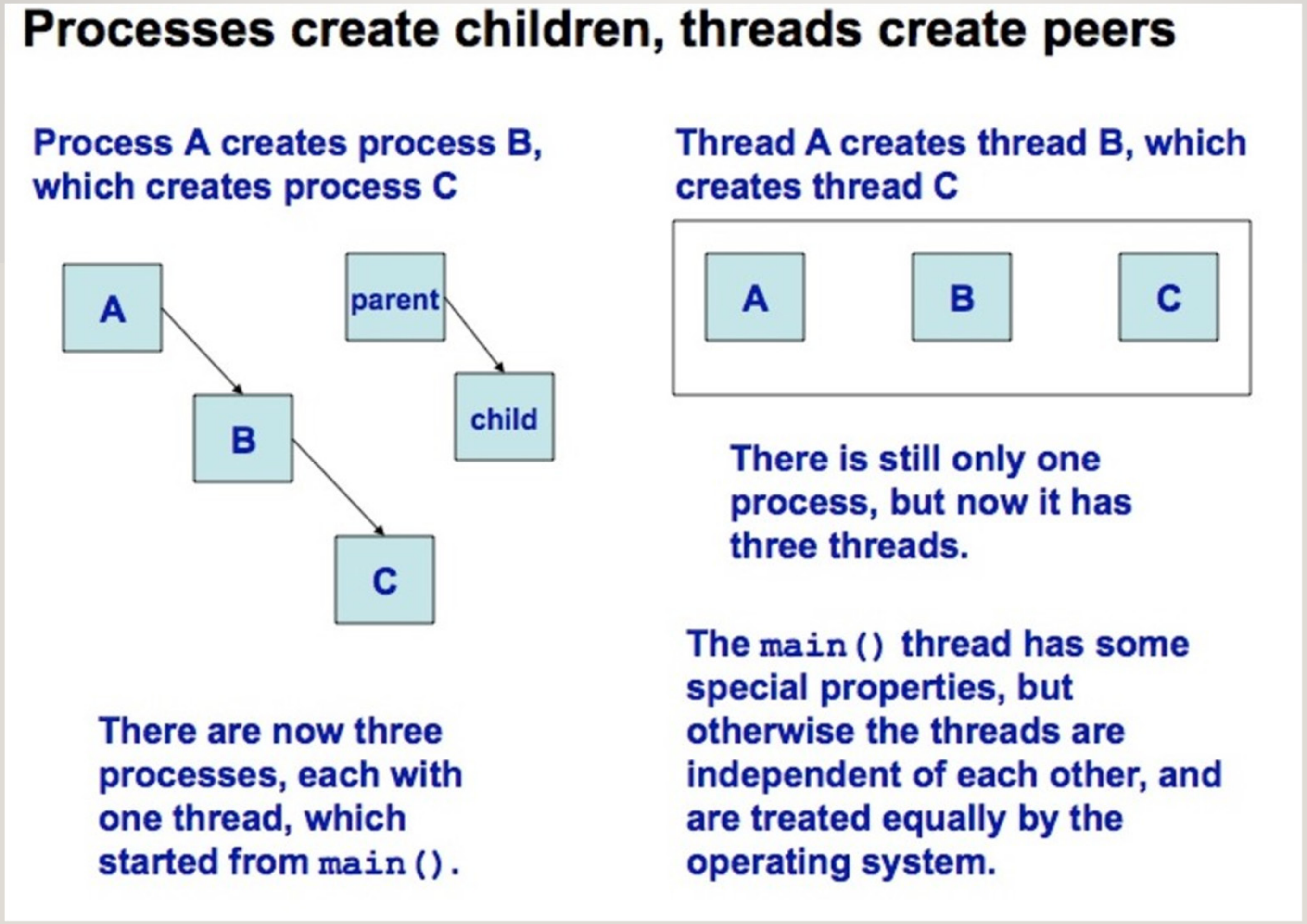
```
Value of x: 3 and process id: 2311
Value of x: 3 and process id: 2316
Value of x: 3 and process id: 2317
Value of x: 3 and process id: 2315
```

Process id is same for all threads. In threads memory is shared among all threads, thus we have the value of x incremented in steps (sleep in the code ensures ++x is completed first). We have two threads and a main process.

Whereas in case of fork, the memory section is not shared, it clones into a new process (diff process ids), thus each process resulting from fork is working on the data section specific to them. Thus the value of x is not taken incrementally.

Since 2 forks, we will have 2^2 = 4 processes (3 child process and 1 parent process)

Processes vs. Threads



Processes create children, threads create peers

Process A creates process B, which creates process C

Thread A creates thread B, which creates thread C

There is still only one process, but now it has three threads.

There are now three processes, each with one thread, which started from `main()`.

The `main()` thread has some special properties, but otherwise the threads are independent of each other, and are treated equally by the operating system.

Source: https://www.cse.psu.edu/~deh25/cmpsc311/Lectures/Threads/Threads.html

fork()

Source: https://www.geeksforgeeks.org/fork-system-call/

Fork System call: It takes no parameters and returns an integer value. Below are different values returned by fork().

Negative Value: creation of a child process was unsuccessful.

Zero: Returned to the newly created child process.

Positive value: Returned to parent or caller. The value contains process ID of newly created child process.

The fork() function shall create a new process. The new process (child process) shall be an exact copy of the calling process (parent process) except as detailed below:

A process shall be created with a single thread. If a multi-threaded process calls fork(), the new process shall contain a replica of the calling thread and its entire address space, possibly including the states of mutexes and other resources.

The functions defined in the pthreads library include:

pthread_create: used to create a new thread

Syntax:

```
int pthread_create(pthread_t * thread,
                   const pthread_attr_t * attr,
                   void * (*start_routine)(void *),
                   void *arg);
```

Parameters:

thread: pointer to an unsigned integer value that returns the thread id of the thread created.

attr: pointer to a structure that is used to define thread attributes like detached state, scheduling policy, stack address, etc. Set to NULL for default thread attributes.

start_routine: pointer to a subroutine that is executed by the thread. The return type and parameter type of the subroutine must be of type void *. The function has a single attribute but if multiple values need to be passed to the function, a struct must be used.

arg: pointer to void that contains the arguments to the function defined in the earlier argument

pthread_exit: used to terminate a thread.
Syntax:

```
void pthread_exit(void *retval);
```

Parameters: This method accepts a mandatory parameter retval which is the pointer to an integer that stores the return status of the thread terminated. The scope of this variable must be global so that any thread waiting to join this thread may read the return status.

pthread_join: used to wait for the termination of a thread.
Syntax:

```
int pthread_join(pthread_t th,
                 void **thread_return);
```

Parameter: This method accepts following parameters:

th: thread id of the thread for which the current thread waits.
thread_return: pointer to the location where the exit status of the thread mentioned in th is stored.

pthread_self: used to get the thread id of the current thread.
Syntax:

```
pthread_t pthread_self(void);
```

pthread_equal: compares whether two threads are the same or not. If the two threads are equal, the function returns a non-zero value otherwise zero.

Source: https://www.geeksforgeeks.org/thread-functions-in-c-c/

Syntax:

```
int pthread_equal(pthread_t t1,
                  pthread_t t2);
```

Parameters: This method accepts following parameters:

t1: the thread id of the first thread
t2: the thread id of the second thread

pthread_cancel: used to send a cancellation request to a thread

Syntax:

```
int pthread_cancel(pthread_t thread);
```

Parameter: This method accepts a mandatory parameter thread which is the thread id of the thread to which cancel request is sent.

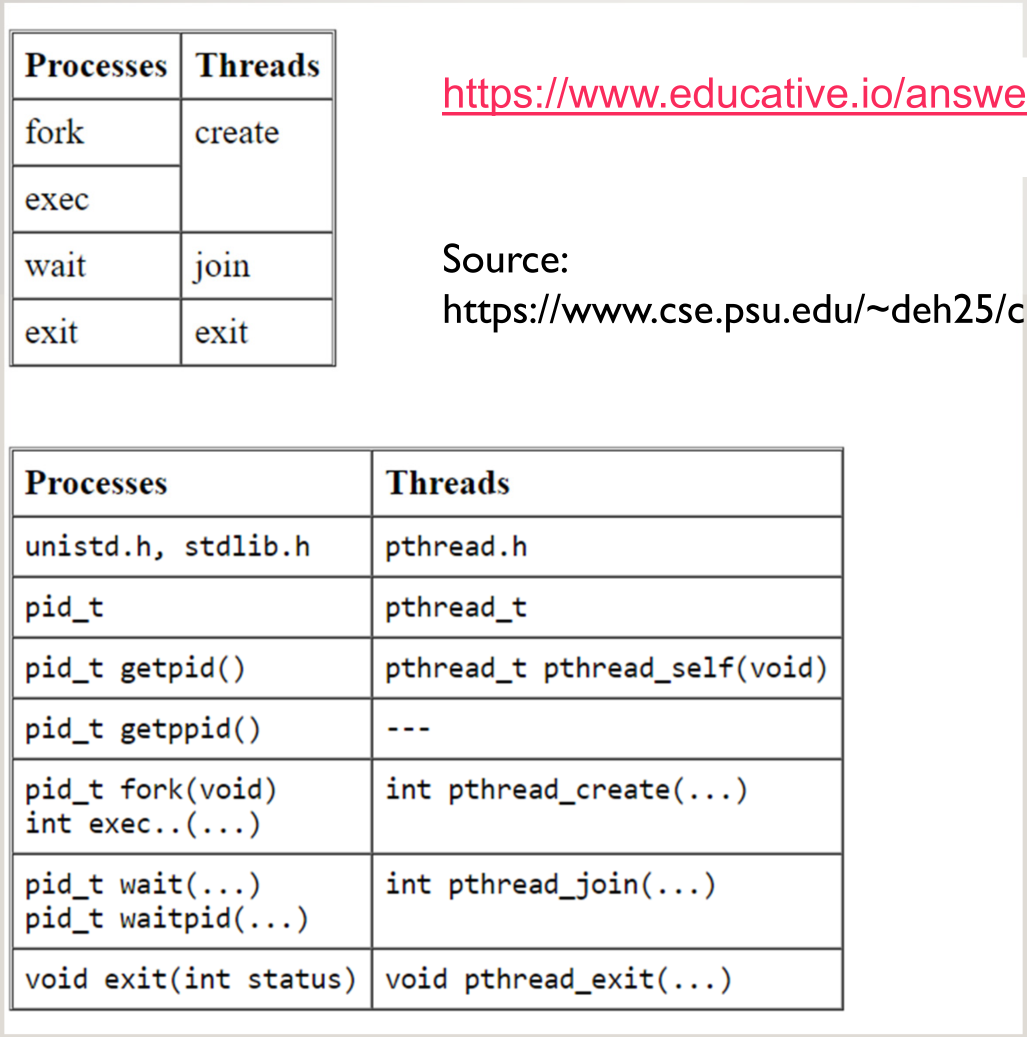
pthread_detach: used to detach a thread. A detached thread does not require a thread to join on terminating. The resources of the thread are automatically released after terminating if the thread is detached.

Syntax:

```
int pthread_detach(pthread_t thread);
```

Parameter: This method accepts a mandatory parameter thread which is the thread id of the thread that must be detached.

Processes vs. Threads



Processes	Threads
fork	create
exec	
wait	join
exit	exit

Processes	Threads
`unistd.h, stdlib.h`	`pthread.h`
`pid_t`	`pthread_t`
`pid_t getpid()`	`pthread_t pthread_self(void)`
`pid_t getppid()`	`---`

`pid_t fork(void)`
`int exec..(...)`
	`int pthread_create(...)`

`pid_t wait(...)`
`pid_t waitpid(...)`
	`int pthread_join(...)`

`void exit(int status)`	`void pthread_exit(...)`

https://www.educative.io/answers/how-to-create-a-simple-thread-in-c

Source:
https://www.cse.psu.edu/~deh25/cmpsc311/Lectures/Threads/Threads.html