1. Prerequisites

1.1. Must be able to use basic technology

1.2. Must have mastery of computer concepts.

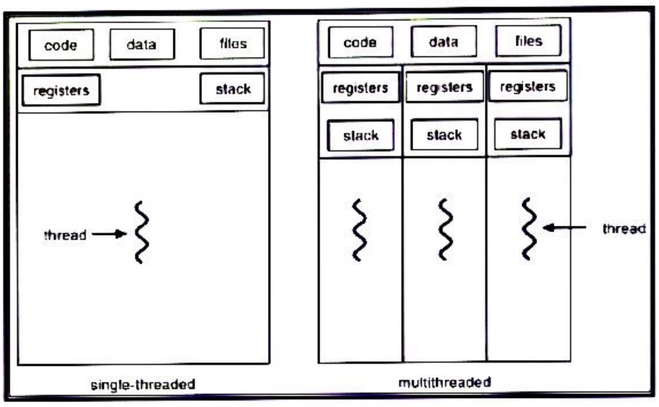
1.3. Must know data structures

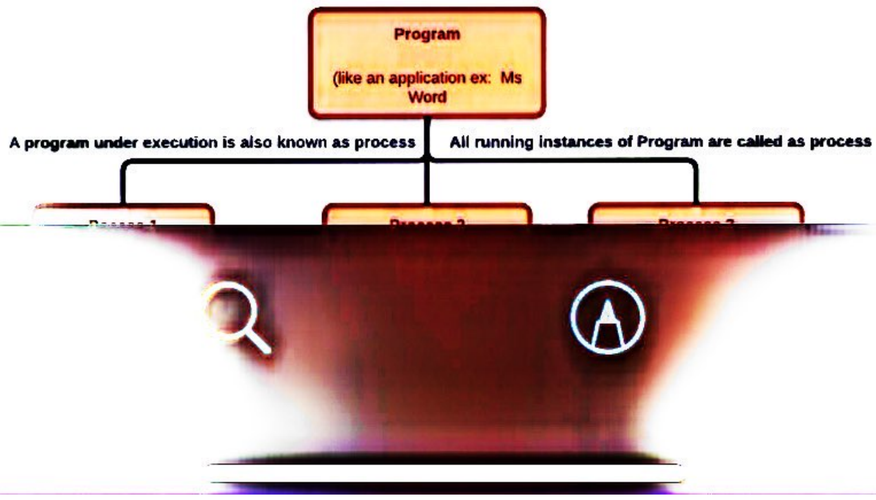
1.4. Must have basic programming skills

1.5. Must be able to develop algorithms

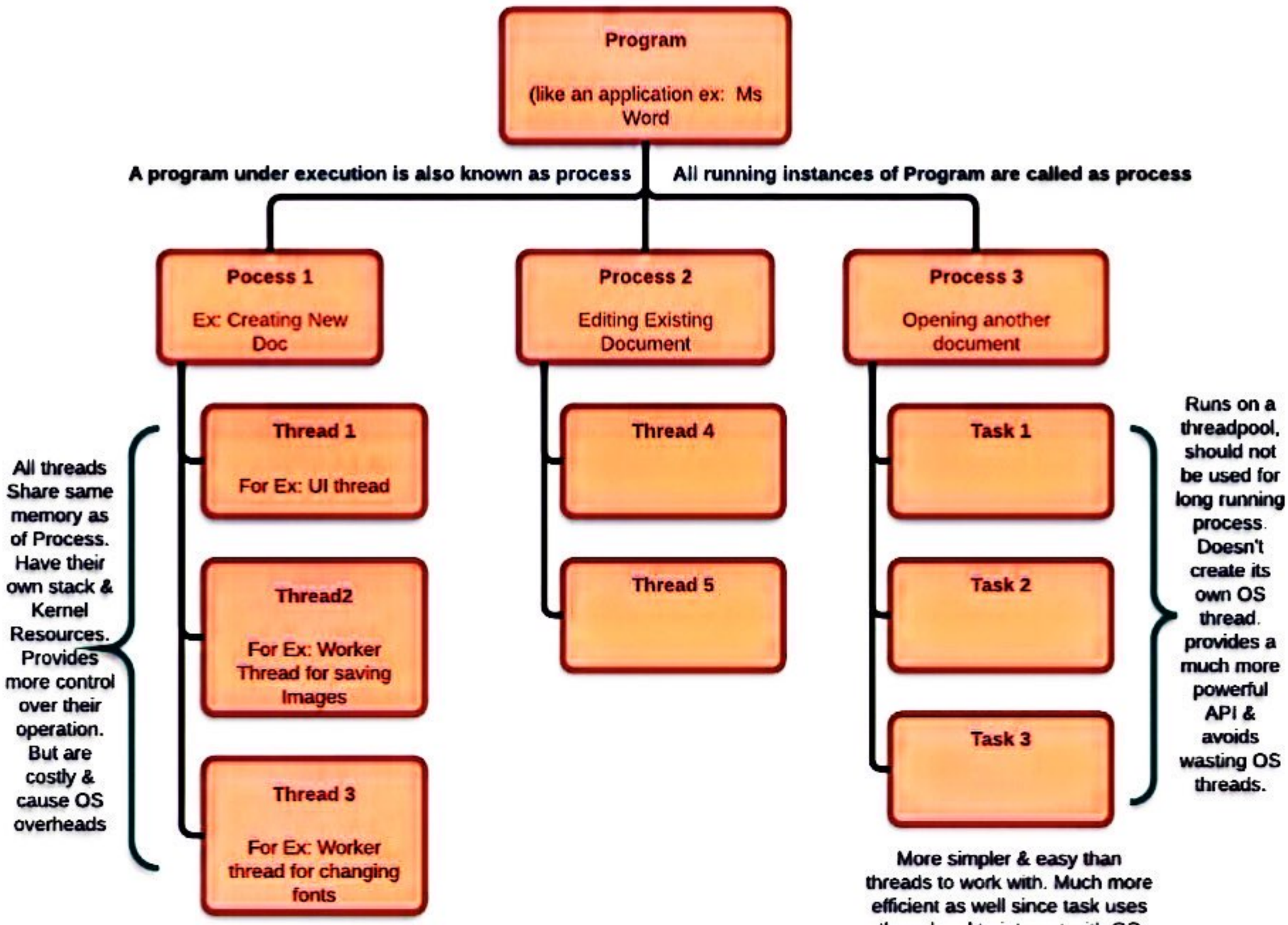
2. What is Thread?

Thread means "thread" in English. In computer science, its meaning is threads. In computer science, processes are used to mean the grouping of tasks that the processor will do. In other words, when you run a program, it is considered to be running on the processor as a process. Threads are threads. It can be thought of as a small job for the processor to do. Threads can be viewed as lightweight processes. They often run underneath processes. Processes can have more than one thread. To threads: "task","pod"





Scanned with CamScanner



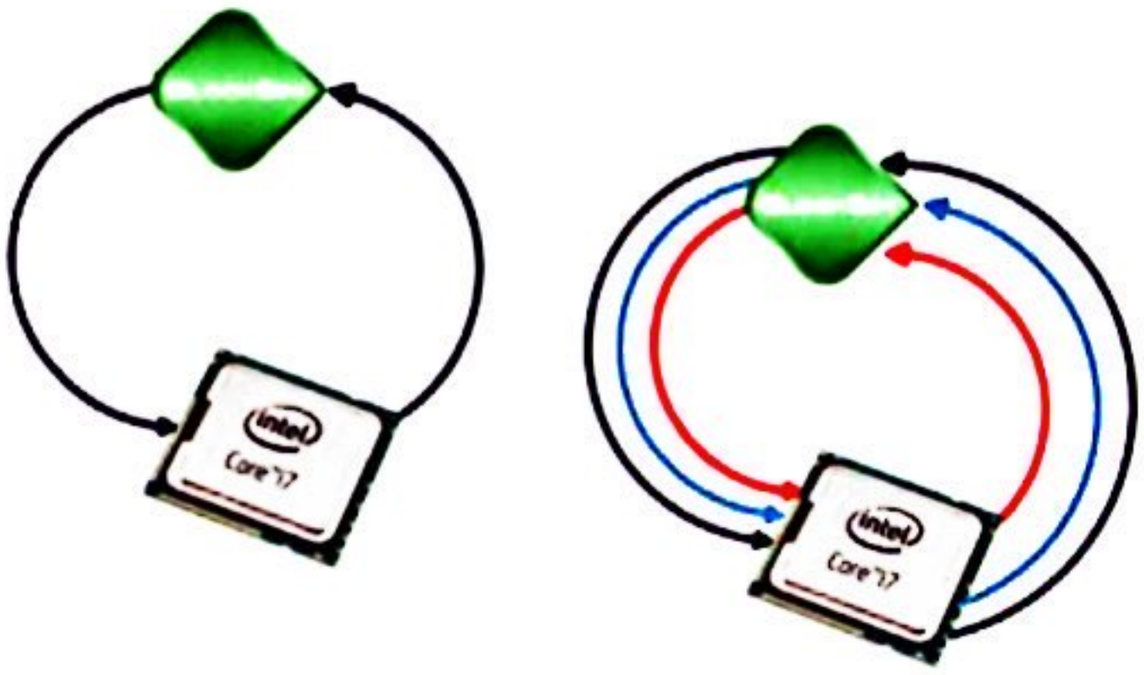
3. What is a thread used for?

Thanks to threads, a job can be divided into threads and processed. The advantage of this is that a job can be processed on different processor cores. These processor cores may be in the same computer or in different computers.

Another point of view: While a process is doing what it needs to do, it cannot skip the queue and must continue its work. However, it would be a very impressive system to be able to do the work that needs to be done while the program continues to flow. We can achieve this with Threads. We send the small job that the main process needs to do to the processor through a thread. Thus, the main process can continue its main

Scanned with CamScanner

work without waiting for the minor job to be finished.



Sometimes a job needs to be done continuously. Like endless loops. For example, constantly listening to a port, waiting for a key from the keyboard, or receiving a signal from another program. Here, if the waiting-listening process is done by the main process, the program will not be able to continue its flow and will focus on this process until it gets a result. Instead, this process can be transferred to the thread and synchronized to communicate with the main process when it is completed.

4. Proses vs Thread

Threads share code, global and heap areas, while Processes have these areas only for themselves. Even if two processes arise from the same process, the PCB is copied when first created and then they continue from separate PCBs. The same is not the case with threads. From the moment two threads are created until they are completely finished, they can access the resources of the process in which they are created and make changes. In this way, they communicate more with each other.

Proses	Thread
The process is heavy.	Thread is lightweight.
It has resource allocation and protection units.	It has a processor usage unit.
The cost of creation is high.	The cost of creation is low.
Processing is slow.	It is fast to process.
They use the same memory space.	They use the same memory space.
Processor resource switching (context switch) is costly.	Processor resource switching (context switch) is cheap.
A process can have many threads.	A thread can belong to only one process.
Cannot share file descriptors.	Shares file descriptors.

Scanned with CamScanner

Cannot share file descriptors.	Shares file descriptors.

Signals are for yourself only.	Can share signals.

5. Creating Threads

Unlike other advanced programming languages, C language has many thread creation and management structures. In this lesson we will use the "pthread.h" library. The Pthread library also includes the necessary support for mutex and conditional operations required for synchronization operations.

When you use the "pthread.h" library, you must add the "-l pthread" section to the compilation code at compile time. The letter "l" here refers to the library parameter.

Write the Prog31 Code.

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
void *printme()
{
  printf("merhaba dunya\n");
  return NULL;
}
int main()
{
  pthread_t tcb;
  void *status;

  if (pthread_create(&tcb, NULL, printme, NULL) != 0) {
    perror("thread olusturulamadi");
    exit(1);
  }
  if (pthread_join(tcb, &status) != 0) { perror("pthread_join"); exit(1); }
  return 0;
}
```

```
insect@insect-Lenovo-G50-80: ~/OpSis/prog31
insect@insect-Lenovo-G50-80:~/OpSis/prog31$ gcc main.c -o prog31 -l pthread
insect@insect-Lenovo-G50-80:~/OpSis/prog31$ ./prog31
Hello World!
insect@insect-Lenovo-G50-80:~/OpSis/prog31$
```

In the code part, first a variable of type "pthread_t" was created to create threads. This variable stands for TCB (thread control block) of the thread. Then, it created a thread object with the "pthread_create" function. In this function, "null" places are unimportant. The last parameter is the arguments to be passed to thread functions. The thread TCB variable should be given as a reference in the first field, and the function that the thread will run should be given in the third field. With this function, the thread started to work. Finally, the "pthread_join" function was called. A variable was given to this function for the TCB of the thread and its status information.

Scanned with CamScanner

```c

void *printme(void *ip)
{
 int *i;

 i = (int *) ip;
 printf("Merhaba.  Ben Thread:  %d\n", *i);
 return NULL;
}

int main()
{
 int i, vals[4];
 pthread_t tids[4];
 void *retval;
 for (i = 0; i < 4; i++) {
   vals[i] = i;
   pthread_create(tids+i, NULL, printme, vals+i);
 }

 for (i = 0; i < 4; i++) {
   printf("tid ile join oluyor %d\n", i);
   pthread_join(tids[i], &retval);
   printf("tid ile join oldu %d\n", i);
 }
 return 0;
}
```

```
insect@insect-Lenovo-G50-80: ~/OpSis/prog32
insect@insect-Lenovo-G50-80:~/OpSis/prog32$ gcc main.c -o prog32 -l pthread
insect@insect-Lenovo-G50-80:~/OpSis/prog32$ ./prog32
Merhaba.  Ben Thread:  1
Merhaba.  Ben Thread:  0
Merhaba.  Ben Thread:  2
Merhaba.  Ben Thread:  3
tid ile join oluyor 0
tid ile join oldu 0
tid ile join oluyor 1
tid ile join oldu 1
tid ile join oluyor 2
tid ile join oldu 2
tid ile join oluyor 3
tid ile join oldu 3
```

Scanned with CamScanner

The thread parameter is of type pthread_t and must be defined beforehand. The resulting thread will always be accessible with this reference.

The attr parameter shows rules such as scheduling policy, stack size, detach policy, etc., which are set by thread-specific functions starting with pthread_attr_.

start_routine indicates the function to be run by the thread.

arg represents a general data structure cast into void* that will be passed to the function to be executed by the thread.

6. Joinable, Detachable Thread

If a thread is returned from the main() function in an application, all threads are terminated and all used resources are returned to the system.

Likewise, if any thread is exited with a command such as exit(), all threads will be terminated.

With the pthread_join function, we can wait for a thread to terminate. The thread on which this function is used will be blocked until the thread that is expected to terminate terminates.

Even if normal (joinable) threads are terminated, if they are not joined with pthread_join, the resources they use will not be given back from the system, even if they are not rescheduled by the CPU.

Sometimes there may be situations where joining with pthread_join is not meaningful and it is not possible to predict when the thread will terminate.

In this case, we can ensure that all resources are automatically returned to the system at the point the thread returns.

To ensure this, we need to start the relevant threads with DETACHED status.

When starting a thread, the DETACH status can be specified via thread attribute values or with the pthread_detach function:int pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

int pthread_detach(pthread_t thread);

Prog33 kodunu yazınız ve çalıştırınız.

Scanned with CamScanner

```c
#include <fcntl.h>
char *convert(int number)
{
void *printme()
{
    int fd= open("dosya.txt", O_CREAT |O_WRONLY | O_APPEND,0777  );
    //perror("");
    for (int var = 0; var < 100; ++var) {
       printf("%d.\n",var);
       char * buffer=convert(var);
       size_t n= strlen(buffer);
       write(fd,buffer,n);
      // perror("");
       sleep(1);
    }
    fclose(fd);
    return NULL;
}

int main()
{
  pthread_t tcb;
  pthread_attr_t attr;
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED);

  if (pthread_create(&tcb, NULL, printme, NULL) != 0) {
    perror("thread olusturulamadi\n");
    exit(1);
  }
   printf("Thread OluÅŸtu\n");
  for (int var = 0; var < 3; ++var) {
    sleep(1);
  }
printf("For bitti\n");
pthread_detach(tcb);
printf("Thread Serbest BÄ±rakÄ±ldÄ±\n");
for (int var = 0; var < 4; ++var) {
  sleep(1);
}
  return 0;
}
```

```
insect@insect-Lenovo-G50-80: ~/OpSis/prog33
insect@insect-Lenovo-G50-80:~/OpSis/prog33$ ./prog33
Thread Oluştu
0.
1.
2.
3.
4.
5.
6.
7.
8.
9.
For bitti
Thread Yok Edildi
insect@insect-Lenovo-G50-80:~/OpSis/prog33$
```

Scanned with CamScanner

7. Thread Cancelling

A thread can be canceled from somewhere else. For this, the TCB value must be known.

int pthread_cancel (pthread_t thread);

Write the Prog 34 code and run it.

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

void *printme()
{
    for (int var = 0; var < 100; ++var) {
       printf("%d.\n",var);
       sleep(1);
    }
    return NULL;
}

int main()
{
  pthread_t tcb;
  pthread_attr_t attr;
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED);
  if (pthread_create(&tcb, &attr, printme, NULL) != 0) {
    perror("thread olusturulamadi\n");
    exit(1);
  }
   printf("Thread OluÅŸtu\n");
  for (int var = 0; var < 10; ++var) {
    sleep(1);
  }
printf("For bitti\n");
  pthread_cancel(tcb);
printf("Thread iptal edildi\n");

for (int var = 0; var < 10; ++var) {
  sleep(1);
}
  return 0;
}
```

Scanned with CamScanner

```
insect@insect-Lenovo-G50-80: ~/OpSis/prog34
insect@insect-Lenovo-G50-80:~/OpSis/prog34$ ./prog34
Thread Oluştu
0.
1.
2.
3.
4.
5.
6.
7.
8.
9.
For bitti
10.
Thread iptal edildi
insect@insect-Lenovo-G50-80:~/OpSis/prog34$
```

8. Exit() vs Pthread_exit()

When the exit() command comes in a C program, the program is terminated. In addition, the exit() command can come from any thread, which closes the entire program. But sometimes we may just want the thread to exit and we will use the pthread_exit() command for this. This command only terminates that thread.

Scanned with CamScanner

Type the prog 35 command and run it.

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
void *printme1()
{
    for (int var = 0; var < 10; ++var) {
        printf("1>> %d.\n",var);
        sleep(1);
    }
  return NULL;
}

void *printme2()
{
    for (int var = 0; var < 10; ++var) {
        printf("2>> %d.\n",var);
        if(var==4)
        {
            pthread_exit(0);
            //exit(0);
        }
        sleep(1);
    }
  return NULL;
}
int main()
{
  pthread_t tcb1;
   pthread_t tcb2;
  void *status;

  if (pthread_create(&tcb1, NULL, printme1, NULL) != 0) {
    perror("thread olusturulamadi");
    exit(1);
  }
  if (pthread_create(&tcb2, NULL, printme2, NULL) != 0) {
    perror("thread olusturulamadi");
    exit(1);
  }

  if (pthread_join(tcb1, &status) != 0) { perror("pthread_join"); exit(1); }

  if (pthread_join(tcb2, &status) != 0) { perror("pthread_join"); exit(1); }
   printf("Bitti\n");
  return 0;
}
```

```
insect@insect-Lenovo-G50-80: ~/OpSis/prog35
insect@insect-Lenovo-G50-80:~/OpSis/prog35$ ./prog35
1>> 0.
2>> 0.
2>> 1.
1>> 1.
1>> 2.
2>> 2.
2>> 3.
1>> 3.
1>> 4.
2>> 4.
insect@insect-Lenovo-G50-80:~/OpSis/prog35$
```

In this program, type pthread_exit() instead of exit() and run it after compiling it again.

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
void *printme1()
{
    for (int var = 0; var < 10; ++var) {
        printf("1>> %d.\n",var);
        sleep(1);
    }
  return NULL;
```

Scanned with CamScanner

In this program, type pthread_exit() instead of exit() and run it after compiling it again.

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
void *printme1()
{
    for (int var = 0; var < 10; ++var) {
       printf("1>> %d.\n",var);
       sleep(1);
    }
  return NULL;
}

void *printme2()
{
    for (int var = 0; var < 10; ++var) {
       printf("2>> %d.\n",var);
       if(var==4)
       {
            //pthread_exit(0);
           exit(0);
       }
       sleep(1);
    }
  return NULL;
}
int main()
{
  pthread_t tcb1;
   pthread_t tcb2;
  void *status;

  if (pthread_create(&tcb1, NULL, printme1, NULL) != 0) {
    perror("thread olusturulamadi");
    exit(1);
  }
  if (pthread_create(&tcb2, NULL, printme2, NULL) != 0) {
    perror("thread olusturulamadi");
    exit(1);
  }

  if (pthread_join(tcb1, &status) != 0) { perror("pthread_join"); exit(1); }

  if (pthread_join(tcb2, &status) != 0) { perror("pthread_join"); exit(1); }
   printf("Bitti\n");
  return 0;
}
```

```
insect@insect-Lenovo-G50-80: ~/OpSis/prog35
insect@insect-Lenovo-G50-80:~/OpSis/prog35$ ./prog35
1>> 0.
2>> 0.
1>> 1.
2>> 1.
2>> 2.
1>> 2.
2>> 3.
1>> 3.
2>> 4.
1>> 4.
1>> 5.
1>> 6.
1>> 7.
1>> 8.
1>> 9.
Bitti
insect@insect-Lenovo-G50-80:~/OpSis/prog35$
```

9. Referances

Scanned with CamScanner