

Write a LINUX program to model space exploration. Five space-ships are exploring new planets looking for five parts of an alien device. Each ship has been assigned to find one part of the device. Each spaceship will travel to a distant star system, search for its part, return to base, and resupply. If the search is successful, the space ship will signal the home base that it has found its part and will make no more trips. Once all parts have been found, the home base will signal the master controller, and both the master controller and the home base will terminate.

The program will have seven threads, a master controller thread, a Home-Port thread, and five Space-Ship threads. The master controller thread will start all other threads. Before starting the threads, it will use the *settimer* and *signal* system calls to set up a periodic timer that will issue a timer signal every 250 milliseconds. The main thread will set up a global variable that contains the current time. The current time will start with zero. Every time a signal is received, the current time will be incremented by 1. The signal processing routine is responsible for timing the six other threads. Each thread will have a global variable that indicates whether it is waiting. If it is waiting, the global variable will be greater than zero. When the timer signal routine executes, it will check each of the five (six?) global variables to see if any are greater than zero. Any that are greater than zero are decremented by one. When a variable is decremented to zero, a condition variable is signaled to wake up the corresponding process. Use a single mutex variable for the entire collection of timer variables. Each ship should print a message each time it changes state. The states are traveling, searching, resupplying and retired. Each ship should print a message when it finds its object. (Note that a ship must return to base before entering the retired state.) The main base should print a message once all parts have been found.

Use global variables and condition variables to signal the home base when a part has been found. There will be one global variable for each part and one condition variable for each part. The home base can wait on each part serially, regardless of the order in which they are found. If a ship finds a part, but the home base is not waiting for that part, then set a global variable indicating that the part has been found. The home base will not wait on a part if the part has already been found. Use a single mutex to control access to the list of found parts.

Use a condition variable to signal the master controller thread that all five parts have been found.

The ships have the following travel times, search times, resupply times and probability of. The travel time to a star system always equals the return time from that star system.

Ship	Travel time	Search Time	Success Probability	Resupply Time
1	1-3 units	1-3 units	12%	1-2 units
2	1-2 units	1-3 units	16%	1-2 units
3	2-5 units	3-5 units	24%	3-4 units
4	1-5 units	3-5 units	30%	3-4 units
5	3-7 units	5-7 units	40%	5-7 units

To compute a random number from x to y use the following expression.

```
MyVar = (rand( )%(y-x+1))+x;
```

Thus to compute a random number from 5 to 7, use the following formula:

```
MyVar = (rand( )%3)+5;
```

To determine success given a probability P, use the following conditional.

```
if ((rand( )%100) < P)
{
    Success = true;
}
```

So for 12% success, use the following conditional.

```
if ((rand( )%100) < 12)
{
    Success = true;
}
```

Set up the timer as follows:

```
#include <sys/time.h>
#include <signal.h>

void SignalRoutine(int)
{ // increment global variables, decrement timer variables
}

// in main
signal(SIGALRM,SignalRoutine);
struct itimerval = {{0,250000},{0,250000}};
setitimer(ITIMER_REAL,&timeval,NULL);
```

Create a thread like this:

```
pthread_t  MyThread;
...
pthread_create(MyThread,NULL,MyRoutine,Args);
...

void *MyRoutine(void *Args)
{
}
```

Exit a thread like this:

```
pthread_exit(NULL);
```

Declare mutex variables like this:

```
pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;
```

The mutex versions of wait and signal are:

```
pthread_mutex_lock(mymutex);  
pthread_mutex_unlock(mymutex);
```

Declare condition variables like this:

```
pthread_cond_t mycondvar = PTHREAD_COND_INITIALIZER;
```

Wait on a condition like this:

```
pthread_mutex_lock(mymutex);  
... (check global variables, set global variables)  
pthread_cond_wait(mycondvar,mymutex);  
...  
pthread_mutex_unlock(mymutex);
```

Signal a condition like this:

```
pthread_mutex_lock(mymutex);  
... (check global variables)  
pthread_cond_signal(mycondvar);  
...  
pthread_mutex_unlock(mymutex);
```

Best pthreads website:

```
http://www.llnl.gov/computing/tutorials/workshops/workshop/pthreads/MAIN.html
```

Turn in: printout of the code, printout of all messages produced in a run, disk or CD with all code.