These are the methods, variables and definitions that you may use throughout home and class exercises :

Definitions:

```
#define INIFINITE_TIMEOUT 0xFFFFFFFF
#define POLICY_FIFO       0x1
#define POLICY_PRIORITY   0x2
#define BLOCK_IF_FAILE    0x1
#define ERROR_IF_FAILE    0x2
#define MAX_PRIORITY          50
#define MIN_PRIORITY          250
```

Variables:

Sem[100], SemA, SemB, SemC, SemD, TimeoutMilliSeconds, MsgQ[100], MsgQ1, MsgQ2, MsgQ3, MsgQ4, task1, task2, task3, task4, task[100]

Methods:

Mutex:

SemMCreate (&Sem, policy), SemMTake (&Sem, timeoutvalue, shouldBlockIfFaile), SemMGive (&Sem)

Inversion Safe Mutex

SemMPISafeCreate (&Sem), SemMPISafeTake (&Sem, timeoutvalue, shouldBlockIfFaile), SemPISafeGive(&Sem)

Binary Semaphores:

SemBCreate (&Sem, initialHoldingCount, policy), SemBTake (&Sem, timeoutvalue, shouldBlockIfFaile), SemBGive (&Sem)

Counting Semaphores:

SemCCreate (&Sem, initialHoldingCount, maximumAllowedHoldings, policy), SemCTake (&Sem, timeoutvalue, shouldBlockIfFaile), SemCGive (&Sem)

Message Queues:

MsgQCreate(&MsgQ, elementSize, NumOfElements, policy), MsgQSend (&MsgQ, inbuffer, timeoutvalue, shouldBlockIfFaile), MsgQReceive(&MsgQ, outBuffer, timeoutvalue)

Tasks:

TaskCreate (&Task, priority), TaskDelay (timeoutvalue), TaskSetPriority (&Task, &NewPriority)
TaskSuspend(&Task), TaskResume(&Task), TaskLock(), TaskUnlock()

Block-IO:

Block (milli-seconds)

Interrupts:

DisableIntr (), EnableIntr ()

Busy : Hold_CPU_For (milli-seconds), UseResource(milli-seconds)

Please assume that the following operations take 1 mill-second: SemTake, SemGive, TaskCreate. The other operations take the time provided as a parameter.

Q1-

You have a system that consists of 4 tasks: Task1, Task2, Task3, Task4. Task1 is the first running task and its priority is 50.
Please provide a time-table:

```
T1 ()
{
        SemCCreate (&SemA, 2, 2, POLICY_FIFO)
        TaskCreate (&T2, 100);
        TaskCreate (&T3, 150);
        TaskCreate (&T4, 200);
}

T2 ()
{
        Hold_CPU_For (10)
        Block (20)
        SemCTake (&SemA, INFINITE_TIMEOUT, BLOCK_IF_FAILE)
        UseResource (8)
        SemCGive (&SemA)
        Hold_CPU_For (1)
}

T3 ()
{
        Hold_CPU_For (8)
        SemCTake (&SemA, INFINITE_TIMEOUT, BLOCK_IF_FAILE)
        UseResource (1)
        Block (40)
        UseResource (3)
        SemCGive (&SemA)
        While (1);
}

T4 ()
{
        Hold_CPU_For (8)
        SemCTake (&SemA, INFINITE_TIMEOUT, BLOCK_IF_FAILE)
        UseResource (9)
        SemCGive (&SemA)
        Hold_CPU_For (2)
}
```

A1

Task	Time	Sem	Priority	Transition
T1	4	A:S=2	50	Compl
T2	10		100	IO(20)
T3	10	A:S--	150	IO(40)
T4	10	A:S--	200	Preempted
T2	1	A :SWQ=1	100	BlockOnSem
T4	9	A:SWQ=0	200	SemReleased-Preempted
T2	10	A:S++	100	Compl
T4	2		200	Compl
IDLE TASK	8		250	Preempted
T3	Forever	A:S++	150	

Q2-
You have a system that consists of 4 tasks: Task1, Task2, Task3, Task4. Task1 is the first running task and its priority is 50.
Please provide a time-table:

```
T1 ()
{
        SemCCreate (&SemA, 0, 2, POLICY_FIFO)
        TaskCreate (&T2, 100);
        TaskCreate (&T3, 150);
        TaskCreate (&T4, 200);
}

T2 ()
{
        Hold_CPU_For (1)
        SemCTake (&SemA, INFINITE_TIMEOUT, BLOCK_IF_FAILE)
        Hold_CPU_For (1)
}

T3 ()
{
        Hold_CPU_For (1)
        SemCTake (&SemA, INFINITE_TIMEOUT, BLOCK_IF_FAILE)
        Hold_CPU_For (1)
}

T4 ()
{
        SemCGive (&SemA)
        SemCGive (&SemA)
        Hold_CPU_For (1)
}
```

A2

Task	Time	Sem	Priority	Transition
T1	4	A:S=0	50	Compl
T2	2	A:SWQ=1	100	BlockOnSem
T3	2	A:SWQ=2	150	BlockOnSem
T4	1	A:SWQ=1	200	SemReleased-Preempted
T2	1		100	Compl
T4	1	A:SWQ=0	200	SemReleased-Preempted
T3	1		150	Compl
T4	1		200	Compl
IDLE TASK	Forever		250	

Q3- Simulate a dead lock

You have a system that consists of 4 tasks: Task1, Task2, Task3, Task4. Task1 is the first running task and its priority is 50.
Please provide a time-table:

```
T1 ()
{
        SemMCreate (&SemA, POLICY_FIFO)
        SemMCreate (&SemB POLICY_FIFO)
        TaskCreate (&T2, 100);
        TaskCreate (&T3, 150);
        TaskCreate (&T4, 200);
}

T2 ()
{
        SemMTake (&SemA, INFINITE_TIMEOUT, BLOCK_IF_FAILE)
        Block (10)
        SemMTake (&SemB, INFINITE_TIMEOUT, BLOCK_IF_FAILE)
        UseResource (40)
        SemMGive (&SemB)
        SemMGive (&SemA)
}

T3 ()
{
        SemMTake (&SemB, INFINITE_TIMEOUT, BLOCK_IF_FAILE)
        SemMTake (&SemA, INFINITE_TIMEOUT, BLOCK_IF_FAILE)

        UseResource (30)

        SemMGive (&SemA)
        SemMGive (&SemB)

}

T4 ()
{
        While (1);
}
```

A3

Task	Time	Sem	Priority	Transition
T1	5	A:S=1 B:S=1	50	Compl
T2	1	A:S--	100	IO(10)
T3	2	B:S-- A :SWQ=1	150	BlockOnSem
T4	8		200	Preempted
T2	1	B :SWQ=1	100	BlockOnSem
T4	Forever		200	

Q4- Simulate priority inversion problem

You have a system that consists of 4 tasks: Task1, Task2, Task3, Task4. Task1 is the first running task and its priority is 50.

Please provide a time-table:

```
T1 ()
{
        SemMCreate (&SemA, POLICY_FIFO)
        TaskCreate (&T2, 100);
        TaskCreate (&T3, 150);
        TaskCreate (&T4, 200);
}

T2 ()
{
        Hold_CPU_For (1)
        Block (10)
        SemMTake (&SemA, INFINITE_TIMEOUT, BLOCK_IF_FAILE)
        UseResource (20)
        SemMGive (&SemA)
}

T3 ()
{
        Hold_CPU_For (1)
        Block (5)
        Hold_CPU_For (70)
}

T4 ()
{
        Hold_CPU_For (1)
        SemMTake (&SemA, INFINITE_TIMEOUT, BLOCK_IF_FAILE)
        UseResource (20)
        SemMGive (&SemA)
        Hold_CPU_For (1)

}
```

A4

Task	Time	Sem	Priority	Transition
T1	4	A:S=1	50	Compl
T2	1		100	IO(10)
T3	1		150	IO(5)
T4	5	A:S--	200	Preempted
T3	4		150	Preempted
T2	1	A:SWQ=1	100	BlockOnSem
T3	65		150	Compl
T4	18	A :SWQ=0	200	SemReleased-Preempted
T2	21	A:S++	100	Compl
T4	1		200	Compl
IDLE TASK	Forever		250	

As you can see – T2 is can't access a critical resource because it's blocked 65 extra milliseconds – just because a mid priority task (T3) that has no relation to this resource has a higher priority than the one of the task that currently owns the mutex. This extra time should be reduced to 0.

Q5-
You have a system that consists of 3 tasks: Task1, Task2, Task3. Task1 is the first running task and its priority is 50.

```
T1 ()
{
        L1: SemBCreate (&SemA, 0, POLICY_FIFO)
        L2: XXXXXXX
        L3: TaskCreate (&T2, 100);
        L4: TaskCreate (&T3, 150);
        L5: XXXXXXX
        L6: XXXXXXX
        L7: Hold_CPU_For (10)

}

T2 ()
{
        L1: Hold_CPU_For (20)
        L2: XXXXXXX
}

T3 ()
{
        L1: XXXXXXX
        L2: Hold_CPU_For (30)
}
```

Our objective is to let T1 reach L7 only after T2 has completed and before T3 reaches L2.
You must not use any delays or change priorities. Use only semaphores – choose the right ones.
5.1 Please replace each instance of XXXXXXX by a valid code.
5.2 Please provide a time-table:

Q6-
You have a system that consists of 2 tasks: Task1, Task2. Task1 is the first running task and its priority is 50.
The program crashed. What was the problem and how should you prevent it using Binary Semaphore.

```
char *ptr;
T1 ()
{
        ptr = malloc (100); /* Assume success */
        TaskCreate (&T2, 100);
        free (ptr);

}

T2 ()
{
        *ptr = 0x10;
}
```

Q7-
This question is almost similar to Q4. Please replace each one of SemMXXX functions by SemMPISafeXXX and provide the time-table.

Q8-

You have a system that consists of 4 tasks: Task1, Task2, Task3, Task4. Task1 is the first running task and its priority is 50.

```
T1 ()
{
        SemBCreate (&SemA, 0, POLICY_FIFO)
        TaskCreate (&T2, 100);
        TaskCreate (&T3, 150);
        TaskCreate (&T4, 200);
        Hold_CPU_For (1)
}

T2 ()
{
        Hold_CPU_For (1)
        Block (10);
        SemBTake (&SemA, INFINITE_TIMEOUT, BLOCK_IF_FAILE)
        Hold_CPU_For (5)
}
T3 ()
{
        Hold_CPU_For (1)
        SemBTake (&SemA, INFINITE_TIMEOUT, BLOCK_IF_FAILE)
        Hold_CPU_For (5)
}
T4 ()
{
        Hold_CPU_For (30)
        SemBGive (&SemA)
        Hold_CPU_For (5)
}
```

8.1 Please provide a time-table.

8.2 Replace POLICY_FIFO by POLICY_PRIORITY. Provide the new time table.

Q9-

You have one global variables var1. There are 60 tasks in the system. Each one of them may call the functions zoo or zoo1. The total concurrent calls to the functions zoo and zoo1 (by concurrent tasks) should not exceed 8. The functions zoo and zoo1 call the functions foo and foo1 respectively. The functions foo() and foo1() can't access Var1 concurrently . The function main (), which was created before – initializes global variables including semaphores. Please add code to the functions main, foo, foo1, zoo and zoo1.

These are the functions:

```
Main ()
{
        XXXXXXX
        XXXXXXX
}

zoo ()
{
        XXXXXXX
        printk ("Hello from zoo\n");
        Foo ()
        XXXXXXX
}
zoo1 ()
{
        XXXXXXX
        printk ("Hello from zoo1\n");
        Foo1 ()
        XXXXXXX
}

foo ()
{
        XXXXXXX
        Var1 += 2;
        XXXXXXX
}
foo1 ()
{
        int a;
        XXXXXXX
        a = Var1;
        XXXXXXX
        printk ("a was read with appropriate value\n");
}
```

Q- Admisión question

There are two tasks T1 and T2 in the system.
The two semaphores that you'll be required to use have already been initialized as follow:

```
SemBCreate (&SemA, 0, POLICY_FIFO)
SemBCreate (&SemB, 0, POLICY_FIFO)
```

The following global variables are initialized, too:

```
Flag = 0;
Global = 0;
```

The priorities of the tasks are not important.
We want to synchronize T1 and T2 in a way that if T1 and T2 completed N loops each (2*N total) - the variable Global should have been incremented only N times.
You must not use any delays or change priorities. Use only semaphores – choose the right ones.
Please replace each instance of XXXXXXX by a valid code.

```
T1 ()
{
        while (1)
        {
                Hold_CPU_For (Unknown Time)
                Block (Unknown Time)
                Hold_CPU_For (Unknown Time)
                Flag |= 0x1 /* Set bit 0 to 1 */
                If (Flag == 0x3)
                {
                        Flag = 0;
                        XXXXXXX
                }
                else
                {
                        XXXXXXX
                        Global ++;
                }
        }
}

T2 ()
{
        while (1)
        {
                Hold_CPU_For (Unknown Time)
                Block (Unknown Time)
                Hold_CPU_For (Unknown Time)
                Flag |= 0x2 /* Set bit 1 to 1 */
                If (Flag == 0x3)
```

```
                {
                        Flag = 0;
                        SemGive (&A)
                }
                else
                {
                        XXXXXXX
                        Global ++;
                }
        }
}
```