1.Race Condition Describe with your own words: What is a race condition?

When two or more procceses want to store data on the same memory address for example.

When two or more processes are in the critical region at the same time.

2.Disabling Interrupts

a)Why is it impossible to achieve Mutual Exclusion via disabling interrupts on a multi-core machine?

because it's only useable for the kernel itself on single-core machines.

deactivating interupts from one core does not prevent that an other core enters the critical region

b)Why is it dangerous to give user processes the power to disable interrupts?

Because a user processes itself is an interrupt and a user process itself could get interrupted for a while.

3.Peterson's Solution

a)Play through the two scenarios of the handout of Peterson's solution. Document how it works.

Scenario 1:

-Process 0 enters the critical region

-All Processes except for Process 0 are blocked

-Process 0 works

-Process 1 waits until Procces 0 has finished

-Process 0 finished the work

-Process 1 enters the critical region

-Process 0 leaves critical region

Scenario 2:

-Process 0 and Process 1 go through the loop nearly simultaneously.

//I assume that 0 is the winner process and 1 the loser process

-Process 1 is the loser process and stuck in the infinity loop until process 0 is interested.

-Process 0 works

-Process 0 is finished with the work and call leave\_region

-Process 0 is not interested any more

-Process 1 can enter the critical region now

b)Play through the scenario which makes the strict alternation approach fail. Document how it fails.

//Process A(Proccess 0) has a high priority and big time slices (for example the scheduler says it may execute 8 codelines in one run)

//Process B(Process 1) has a low priority and small time slices (for example the scheduler says it may execute 2 codelines in one run)

-Process 0 looks if turn != 0, turn = 0 Process 0 can continue and is not stucked in the while

-Process 0 executes 7 further codelines

-Process 0 executes critical\_region();

-Process 0 executes turn= 1;

-Process 0 executes noncritical\_region();

-Process 0 looks 4 times if turn != 0 (while(turn != 0);) and is stuck in the while

-Process 0 executed 8 codelines so process 1 has its turn now

-Process 1 looks if turn != 1,turn = 1 so Procces 1 is not stuck in the while and can continue

-Process 1 executes critical\_region()

-Process 1 executed 2 codelines os Process 0 has its turn now

-Process 0 looks 8 times if turn != 0 (while(turn != 0);), turn = 1, Process 0 is stucked

!!! A lot of time was wasted

that is how it fails the one way

the other way of failure is that if Process 1 would be allowed to execute only one codeline in a turn

then following scenario could happen:

- Procces 1 executes turn = 0;

-Process 1 excuted 1 commandline,so Process 0 has its turn

// Process 0 can go in the critical region even if process 1 is in it too

c)What is the meaning of the variable loser in Peterson's solution? When does it have any effect?

The loser variable stores which process waits for the other process.

It has effect when the infinity loop catches the loser process, the winner proccess

will go into the critical region and won't be stucked in the loop.

d)Extend the given functions such they can handle three processes.

int winner // shared variable

Bool interested[3] // 3 processes only

int getProccess(int process, int number){

int ret = process -number;

if(ret == -1){

return 2;

}

if(ret == -2){

return 1;

}

return ret;

}

void leave\_region(int process)

{

interested[process] = false;

winner=getPRocess(winner, 1);

interested[winner] = true;

}

void enter\_region(int process)

{

int otherOne = getProccess(procces,1);

int otherTwo = getProccess(process,2);

winner = process;

interested[winner] = true;

while (winner != process && interested[winner]) ;

}