Ex. 1 - Simple Questions

1. A clock interrupt during any thread's switching state, namely being blocked or unblocked, may cause unexpected problem.

   Solution: Check queue before clock interrupt; apply the clock interrupt only when no threads are changing states.

2. Yes. `select()` and `pselect()` allow a program to monitor multiple file descriptors, waiting until one or more of the file descriptors become "ready" for some class of I/O operation. If it's not available for a certain operating system, however, we can still use alarm clock to serve a similar role. Specifically, the alarm clock can be used to check the state of a thread before executing a system call. If the system call is blocked, the thread will re-take the control.

Ex2 Monitors

`waituntil` may consume more resources since it will check the value of variable changed every time, while `wait` and `signal` only care about the signal; the process will be awaken with the signal.

Ex3 Race Condition in Bash

The naive code in `ex3_naive.sh` is shown as follows:

```
#!/bin/bash
out=./ex3_naive

if ! test -f "$out"; then
    echo "0" >> $out
fi

for i in {1..50}
do
    operand=$(tail -n 1 $out)
    ((operand++))
    echo $operand >> $out
done
```

We run it both background and foreground through `driver.sh`:

```
#!/bin/bash
for i in {1..20}
do
    ./ex3.sh; ./ex3.sh&
done
```

We observe that the race condition happens after around `51` loops.

We add `flock` to avoid the race condition, which is shown in `ex3_revised.sh`:

```
#!/bin/bash
out=./ex3_revised
if ! test -f "$out";
then
    echo "0" >> $out
fi

for i in {1..50}
do
(
    flock -n -x 33
```

```
    if [ $? -eq 1 ];
    then
        exit;
    fi
    operand=$(tail -n 1 $out)
    ((operand++))
    echo $operand >> $out
) 33>>$out
done
```

The revised driver program:

```
#!/bin/bash
for i in {1..20}
do
    ./ex3_revised.sh; ./ex3_revised.sh&
done
```

Check output `ex3_revised`, and we find the issue has been settled.

Ex4 semaphore

see `./ex4.c` for more detail.