Solutions to the first midterm

COSC 4330/6310
Summer 2013
First question

a) Give an example of a popular operating system using a UNIX or a Linux kernel.

b) Give an example of a real-time process with hard deadlines.
First question

a) Give an example of a popular operating system using a UNIX or a Linux kernel.
   - Mac OS X or Android

b) Give an example of a real-time process with hard deadlines.
First question

a) Give an example of a popular operating system using a UNIX or a Linux kernel.
   - Mac OS X or Android

b) Give an example of a real-time process with hard deadlines.
   - Industrial process control, missile guidance system, ...

c) What is the main disadvantage of *microkernels*?

d) What is the main advantage of *delayed writes*?
First question

c) What is the main disadvantage of microkernels?
   They are slower than other kernel organizations

d) What is the main advantage of delayed writes?
First question

c) What is the main disadvantage of microkernels?
   ➢ They are slower than other kernel organizations

d) What is the main advantage of delayed writes?
   ➢ They reduce the number of disk accesses
   ➢ They return faster
First question

e) Where was the first operating system with a graphical user interface developed?

f) Which event(s) will move a process from the waiting state to the ready state?
First question

e) Where was the first operating system with a graphical user interface developed?

- At Xerox Palo Alto Research Center (Xerox PARC)

f) Which event(s) will move a process from the waiting state to the ready state?
First question

e) Where was the first operating system with a graphical user interface developed?

- **At Xerox Palo Alto Research Center (Xerox PARC)**

f) Which event(s) will move a process from the *waiting state* to the *ready state*?

- Whenever a system request issued by the process completes
Second question

- What would be the main disadvantage of a processor that would not have (3×5 points)
  a) Separate supervisor and user modes?

  b) Memory protection?

  c) Timer interrupts?
Second question

- What would be the *main disadvantage* of a processor that would not have (3×5 points)
  
  a) Separate *supervisor* and *user modes*?
  
  - We could not prevent user programs from directly accessing the disk

  b) *Memory protection*?

  c) *Timer interrupts*?
Second question

- What would be the main disadvantage of a processor that would not have (3×5 points)
  a) Separate **supervisor** and **user modes**?
    - We could not prevent user programs from directly accessing the disk
  b) **Memory protection**?
    - We could not prevent user programs from corrupting the kernel
  c) **Timer interrupts**?
Second question

- What would be the main disadvantage of a processor that would not have (3×5 points)
  a) Separate supervisor and user modes?
    - We could not prevent user programs from directly accessing the disk
  b) Memory protection?
    - We could not prevent user programs from corrupting the kernel
  c) Timer interrupts?
    - We could not prevent user programs from monopolizing the CPU
Third question

- Add the missing code to the following program to ensure it will always print:
  - Child says hello!
  - Parent says hello!
- in that exact order. (3×5 points)
The program

```c
#include <unistd.h>
#include <stdio.h>
void main() {
    int pid;
    if ((pid = fork()) == _____) {
        printf("Child says hello!\n");
        ________________________________;
    } // if
    ________________________________;
    printf("Parent says hello!\n");
} // main
```
The program

```c
#include <unistd.h>
#include <stdio.h>
void main() {
    int pid;
    if ((pid = fork()) == 0) {
        printf("Child says hello!\n");
   } // if
    printf("Parent says hello!\n");
} // main
```
The program

```c
#include <unistd.h>
#include <stdio.h>
void main() {
    int pid;
    if ((pid = fork()) == 0 ) {
        printf("Child says hello!\n");
        _exit(0);  // to terminate the child
    } // if
    printf("Parent says hello!\n");
} // main
```
The program

#include <unistd.h>
#include <stdio.h>
void main() {
    int pid;
    if ((pid = fork()) == 0) {
        printf("Child says hello!\n");
        _exit(0); // to terminate the child
    } // if
    wait(0); // wait first for child completion
    printf("Parent says hello!\n");
} // main
Fourth question

- What does a program do when it receives a signal? (5 points)

- What can we do to change this behavior? (5 points)

- Is it always possible? (5 points)
Fourth question

- What does a process do when it receives a signal? (5 points)
  - **It terminates**
- What can we do to change this behavior? (5 points)
- Is it always possible? (5 points)
Fourth question

- What does a process do when it receives a signal? (5 points)
  - It terminates

- What can we do to change this behavior? (5 points)
  - Process can catch the signal using signal()

- Is it always possible? (5 points)
Fourth question

- What does a process do when it receives a signal? (5 points)
  - It terminates

- What can we do to change this behavior? (5 points)
  - Process can catch the signal using signal(...) system call

- Is it always possible? (5 points)
  - SIGKIL signal cannot be caught
Fifth question

How will the following code fragment affect stdin, stdout and stderr? (3×5 points)
int fda, fdb;

data = open("alpha", O_RDWR | O_CREAT, 0640);
dba = open("beta", O_RDWR | O_CREAT, 0640);
close(0);
dup(fda);
close(1);
dup(fdb);
The code

```c
int fda, fdb;

fda = open("alpha", O_RDWR | O_CREAT, 0640);
fdb = open("beta", O_RDWR | O_CREAT, 0640);
close(0);
dup(fda);  // fda duplicated into stdin
close(1);
dup(fdb);

stdin will read from file "alpha"
```
The code

- **int fda, fdb;**
  
  ```c
  fda = open("alpha", O_RDWR | O_CREAT, 0640);
  fdb = open("beta", O_RDWR | O_CREAT, 0640);
  close(0);
  dup(fda); // fda duplicated into stdin
  close(1);
  dup(fdb); // fdb duplicated into stdout
  ```

- **stdin will read from file "alpha"**
- **stdout will be redirected to file "beta"**
```c
int fda, fdb;

fda = open("alpha", O_RDWR | O_CREAT, 0640);
fdb = open("beta", O_RDWR | O_CREAT, 0640);
close(0);
dup(fda);  // fda duplicated into stdin
close(1);
dup(fdb);  // fdb duplicated into stdout

stdin will read from file "alpha"
stdout will be redirected to file "beta"
stderr will be unchanged
```
Sixth question

- What happens when a user-level thread does a blocking system call? (5 points)

- What can we do to avoid that? (5 points)
Sixth question

What happens when a *user-level thread* does a *blocking system call*? (5 points)

- The CPU scheduler will put the whole process into the waiting state even when other threads are ready to run

What can we do to avoid that? (5 points)
Sixth question

- What happens when a *user-level thread* does a *blocking system call*? (5 points)
  - The CPU scheduler will put the whole process into the waiting state even when other threads are ready to run
- What can we do to avoid that? (5 points)
  - The programmer cannot use blocking system calls
  - Use kernel-supported threads