University of Stuttgart
Institute of Parallel and Distributed Systems
Distributed Systems Department

Winter Term 2017/2018

Operating Systems

Assignment Sheet 1

Available:	Monday, October 23, 2017, 08:00 am
Due Date:	Monday, November 6, 2017, 08:00 am
Presentation:	Friday, November 10, 2017, 11:30-13:00, Room V38.02

General remarks: The intent of the exercises is to help you get more familiar with fundamental concepts of modern operating systems as presented in the lecture.

Work on the problems with your group and hand in a single solution which will be graded for all members of the group. We will review your solution and hand out the graded version in the exercise sessions. Everybody has to present at least once in the sessions. If you hand in a solution but are not able to present it, no points will be given *for the whole exercise*! If we detect any plagiarism in your solution, *the whole group* will not get points *for the whole sheet.*

Please read the *SubmissionGuidlines.pdf* and *InformationOnExercises.pdf* in ILIAS in the Course Material folder. Especially, **be concise** and avoid long answers.

For additional reference, consult the text books cited in the lecture and explore the Web. You can find additional information on lectures and exercises at the lecture homepage. If you have any questions, you may contact us on the e-learning system ILIAS.

Questions

1. (6 points) **Processor and Instruction Cycle**
   (a) In general terms, what are the four distinct actions that a machine instruction can specify? Also explain these actions briefly. (2 points)
   (b) Which registers are used by a processor to exchange data with main memory and I/O devices? Explain briefly (one sentence each) how these registers are used. (2 points)
   (c) For the registers used for data exchange with main memory in part (b), explain, in a step-wise manner, how they are used during an instruction cycle. Assume that interrupts are disabled. (2 points)

2. (5 points) **Memory Hierarchies**

   Lets assume that in a system with a memory hierarchy that consists of a cache, main memory and a disk, a reference to a word takes 1 ns if it is found in the cache. In order to copy a memory word from main memory to cache, 50 ns are needed. Moreover, it takes 10 μs to copy the word from the disk to main memory.
   (a) Explain what happens when a memory word is referenced. (1 point)
   (b) Compute the average time of access to a word. Assume that the probability to find a word in the cache is 0.6 and the probability to find a word in main memory (when it is not in the cache) is 0.4. (2 points)
   (c) Explain how *spatial locality* and *temporal locality* are exploited to improve the hit rate of cache memory. For both locality terms, support your arguments with two examples: one for instructions and one for data. (2 points)

3. (6 points) **Interrupts, DMA and Dispatcher**
   (a) The following pseudocode shows the definition of process A and process B. The processes can be interrupted by an Interrupt Service Routine(ISR) or they can call the sleep function. The pseudocode for the ISR and the sleep function are also given below.

Process A

```
int main()
{
10      some operation;
20  sleep();
30      some operation;
}
```

Process B

```
int main()
{
70      some operation;
80      (Interrupt Raised)
90      some operation;
}
```

Process *A* calls the system function sleep. While in execution of Process *B* at instruction number 80 as shown above an Interrupt is raised in turn invoking the Interrupt service routine(ISR).

ISR

```
interrupt IntServiceRoutine()
{
...
sleep();
...;
}
```

sleep

```
void sleep()
{
...
sleep for 10 seconds
Processor control released;
...
}
```

As shown, the sleep function releases the processor control allowing the scheduler to dispatch the next process. The ISR is a normal interrupt routine but along with a call to the function sleep(). In the above execution of Process *A* and Process *B*, which is better in terms of CPU utilization? Justify your answer. For your explanation, assume that the interrupt handler disables further interrupts.(3 points)

(b) In virtually all systems that include DMA modules, DMA access to main memory is given higher priority than processor access to main memory. Explain why! (1 point)

(c) What is the main optimization goal for a dispatcher? Which approaches do you know to achieve this? What is the tradeoff? (2 points)

4. (8 points) **Implementation Task**

Developing a Shell

In this exercise, you are required to develop a small shell, `gbsh`, which implements some of the features found in typical shells, like the `bash` (Bourne Again Shell) or `csh` (C-Shell).

We will use C language for implementing the shell as it is the standard language for system programming in the Unix environment.

NOTE: It is highly recommended that you use gcc compiler (and not g++ or others).

To get you started, we provide you with an archive file `pracex.tar`, downloadable in the same location as this sheet. The archive contains some basic files. Create folder `gbs-ws1718` in your home directory and extract `pracex.tar` to that folder.

You will be able to implement the problem on a standard Linux system, as installed on the machines in the computer pools of the computer science building
(see `http://www.zdi.uni-stuttgart.de/rechnerpools.html`). If you do not have an account for these pools, you may ask for one at the pool's information desk (Benutzerberatung).

1. In the first part, we will implement some basic shell functionality to get started.

   Some basic files for the shell are contained in folder `gbs-ws1718`, and you should put all other files that you may need in this folder.

   File `gbsh.c` should contain the main code for the shell. You may create other files if you want to split functionality in a modular fashion. The provided `Makefile` will build your program. If you use additional source code files, you need to add them to the `Makefile` in order for them to get built into the executable.

In this problem, implement the following features of the first version of `gbsh`. While implementing these features avoid using the library function system() wherever possible.

(a) **Prompt:** Create a loop that outputs a prompt that awaits user input. Once a command is entered and executed by the user by pressing "return", output the command to `stdout` and await the next user input. The prompt should have the following format:

  `<user>@<host> <cwd> >` (with space at the end)

Here, `<user>` should be your login name, `<host>` the host name you are logged in to, and `<cwd>` the current working directory. Use system calls to retrieve that information and do not hard-code that information in your program (2 Points).

(b) **exit** command: Implement the `exit` command that quits the shell (1 Point).

(c) **pwd** command: Implement the `pwd` command that prints the user's current working directory (1 Point).

(d) To add some mathematical capabilities to your shell, you are required to implement the **sum-max** command which is described below. Please note that you will be required to re-use a modified version of this command in the later exercises. Therefore, you should ensure that your implementation is correct.

The `sum-max` command takes as input two matrices, A and B, and also computes an output matrix C. The matrices A and B have the same size, i.e., m rows and k columns, whereas the output matrix C contains 1 row and k columns. To compute C matrix, the `sum-max` command performs the following 2 steps.

- In the first step, an intermediate matrix D is computed where each element d_{ij} (element at ith row and jth column) is determined as the maximum of the corresponding elements of matrices A and B, i.e., $d_{ij} = max(a_{ij}, b_{ij})$. Note that the size of matrix D is equal to the sizes of matrices A and B, i.e., $m \times k$.
- In the second and final step, matrix C, containing a single row, is computed where each element c_j is determined by the sum of the elements of the jth column of matrix D, i.e., $c_j = \sum_{i=1\ldots m} d_{ij}$.

For your implementation, you can generate random input matrices A and B at run-time with sizes up to a maximum of 5×5. Write a separate code file for the `sum-max` command and add it to the `Makefile` in order to ensure its compilation (if necessary, research the web on how to do it). (3 Points)

(e) For any other command, print the statement "Unrecognized command" to the console and await the next command from the user after printing the above statement (1 Point).