Exam Questions

2019-11-12
KTH Kista - ID1206 Operating Systems

Structure of the Exam

- Questions from the book, lectures, seminars and assignments
- Two Parts -
  - Part 1
    - 5 Questions, 10 points. Fx - 7 points, E - 8 points
    - the student should be able to:
      - explain how multi-threaded processes are structured
      - explain how an operating system can handle several processes at the same time
  - Part 2
    - For higher grades. 5 Pass/Fail questions. where 2 Correct = D, 3 = C, 4 = B, 5 = A
    - the student should be able to:
      - explain how virtualisation of memory is implemented
      - explain how memory management is implemented
      - explain properties for different scheduling algorithms
      - explain properties for different types of process communication
      - explain implementation of more advanced file system.

How to Study for the Exam?

- Read the book
- Go through the lectures
- Do the assignments
  - Processes
  - Virtual memory
  - Concurrency
  - File Systems

- You are allowed to have a self handwritten A4 paper with you

Part 1. Question 1

1 stack or heap [2 points]

What is done in the procedure below and where should `gurka` be allocated? Why? Complete the code so that `gurka` is allocated space.

```
int *tomat(int *a, int *b) {
  // allocate room for gurka


  *gurka = *a + *b;
  return gurka;
}
```

Part 1. Question 1

1 stack or heap [2 points]

What is done in the procedure below and where should `gurka` be allocated? Why? Complete the code so that `gurka` is allocated space.

```
int *tomat(int *a, int *b) {
  // allocate room for gurka

  *gurka = *a + *b;
  return gurka;
}
```

Answer: The function will return a pointer to a `int` that then must be allocated on the heap. The variable `gurka` should point to a heap allocate are large enough to store an `int`. This is achieved by:

```
int *gurka = (int*)malloc(sizeof(int));
```

Part 1. Question 2

2 fork() [2 points]

What is printed when we run the program below, what alternatives exist and why do we get this result?

```
int global = 17;

int main() {
  int pid = fork();
  if(pid == 0) {
    global++;
  } else {
    global++
    wait(NULL);
    printf("global = %d \n", global);
  }
  return 0;
}
```

Part 1. Question 2

2 fork() [2 points]

What is printed when we run the program below, what alternatives exist and why do we get this result?

```
int global = 17;

int main() {
  int pid = fork();
  if(pid == 0) {
    global++;
  } else {
    global++
    wait(NULL);
    printf("global = %d \n", global);
  }
  return 0;
```

Answer: The output will be `global = 18` once. The two processes will have their own copy of the data segment and hence `global`. Since the updates are independent of each other the final result in both cases is 18. Only the mother process will output the result.

Part 1. Question 3

3 `__sync_val_compare_and_swap()` [2 points]

We can implement a spin lock in GCC as shown below. The lock is implemented using a machine instruction that atomically will read the content of a memory location and, if it is equal to our requirement, replace it with a new value. In the implementation below we represent an open lock with the value 0; if the lock is open we write a 1 in the location and return 0, otherwise we return the value found (that then should be 1).

Assume that we use the lock to synchronize two threads on a machine with only one core; what is then the disadvantage that we will have? How could we mitigate the problem?

```
int try(volatile int *mutex) {
  return __sync_val_compare_and_swap(mutex, 0, 1);
}

void lock(volatile int *mutex) {
  while(try(mutex) != 0) { }
}

void release(volatile int *mutex) {
  *mutex = 0;
}
```

Part 1. Question 3

Answer: If a thread takes the lock and is then suspended by the scheduler, the scheduled thread could in the worst case spend its whole allotted time slot spinning. We could yield the CPU, `pthread\_yield()`, to allow the suspended thread to continue its execution.

3 `__sync_val_compare_and_swap()` [2 points]

We can implement a spin lock in GCC as shown below. The lock is implemented using a machine instruction that atomically will read the content of a memory location and, if it is equal to our requirement, replace it with a new value. In the implementation below we represent an open lock with the value 0; if the lock is open we write a 1 in the location and return 0, otherwise we return the value found (that then should be 1).

Assume that we use the lock to synchronize two threads on a machine with only one core; what is then the disadvantage that we will have? How could we mitigate the problem?

```
int try(volatile int *mutex) {
  return __sync_val_compare_and_swap(mutex, 0, 1);
}

void lock(volatile int *mutex) {
  while(try(mutex) != 0) { }
}

void release(volatile int *mutex) {
  *mutex = 0;
}
```

Part 1. Question 4

This is the print out. Explain what is found at the locations indicated by arrows.

```
0x7ffca03d1748      0x3
0x7ffca03d1750      0x7ffca03d1750
0x7ffca03d1758      0xb93d7906926a7d00
0x7ffca03d1760      0x7ffca03d1790        <-------
0x7ffca03d1768      0x55cdac31d78c        <-------
0x7ffca03d1770      0x7ffca03d17d8
0x7ffca03d1778      0x7ffca03d17b0
0x7ffca03d1780      0x1
0x7ffca03d1788      0x2
0x7ffca03d1790      0x7ffca03d17c0
0x7ffca03d1798      0x55cdac31d7c2
0x7ffca03d17a0      0x55cdac31d810
0x7ffca03d17a8      0x12acac31d5f0
0x7ffca03d17b0      0x1
  p: 0x7ffca03d17b0
  back: 0x55cdac31d7c2
```

4 a stack, a bottle and.. [2 points]

You have written the program below to examine what is on the stack.

```
void zot(unsigned long *stop ) {
  unsigned long r = 0x3;
  unsigned long *i;
  for(i = &r;  i <= stop; i++){ printf("%p          0x%lx\n", i, *i);  }
}

void foo(unsigned long *stop ) {
  unsigned long q = 0x2;
  zot(stop);
}

int main() {
  unsigned long p = 0x1;
  foo(&p);
 back:
  printf("  p: %p \n", &p);
  printf("  back: %p \n", &&back);
  return 0;
}
```

Part 1. Question 4

4 a stack, a bottle and.. [2 points]

You have written the program below to examine what is on the stack.

```
void zot(unsigned long *stop ) {
  unsigned long r = 0x3;
  unsigned long *i;
  for(i = &r;  i <= stop; i++){ printf("%p          0x%lx\n", i, *i);  }
}

void foo(unsigned long *stop ) {
  unsigned long q = 0x2;
  zot(stop);
}

t main() {
  unsigned long p = 0x1;
  foo(&p);
 back:
  printf("  p: %p \n", &p);
  printf("  back: %p \n", &&back);
  return 0;
}
```

This is the print out. Explain what is found at the locations indicated by arrows.

```
0x7ffca03d1748      0x3
0x7ffca03d1750      0x7ffca03d1750
0x7ffca03d1758      0xb93d7906926a7d00
0x7ffca03d1760      0x7ffca03d1790
0x7ffca03d1768      0x55cdac31d78c
0x7ffca03d1770      0x7ffca03d17d8
0x7ffca03d1778      0x7ffca03d17b0
0x7ffca03d1780      0x1
0x7ffca03d1788      0x2
0x7ffca03d1790      0x7ffca03d17c0
0x7ffca03d1798      0x55cdac31d7c2
0x7ffca03d17a0      0x55cdac31d810
0x7ffca03d17a8      0x12acac31d5f0
0x7ffca03d17b0      0x1
  p: 0x7ffca03d17b0
  back: 0x55cdac31d7c2
```

Previous stack base pointer (EBP)
<-------
<-------
return address from zot.

Part 1. Question 5

5 library call vs system call [2 points]

An operating system that implements POSIX should provide specified functionality to a user process. Is this provided by system calls, library procedures or a combination of both? Explain the difference between system calls and procedure calls and which parts belong to the operating system.

Part 1. Question 5

5 library call vs system call [2 points]

An operating system that implements POSIX should provide specified functionality to a user process. Is this provided by system calls, library procedures or a combination of both? Explain the difference between system calls and procedure calls and which parts belong to the operating system.

Answer: POSIX is partly provided by system calls and partly be library procedures, they both belong to the operating system. Library procedures are executed in user mode and can thus work without a costly context switch. Since library procedures are more efficient, as much as possible should be implemented using them. Library procedures are however limited in that they do not have access to the global data structures of the operating system and can thus not provide all functionality. Even if not all can be done i user space it is, as in malloc/free, an advantage to do most of the work there.

Part 2. Question 6

6 from one to the ...[P/F]

Assume that we have two programs, `ones` and `add2`, implemented as bellow. The call to `scanf("\%d", \&in)` will read from `stdin` and parse a number that is then stored in `\&in`. The procedure either returns 1, if it manages to read number, or `EOF`. The call to `printf()` will write the number to `stdout`.

```c
/* ones.c */
#include <stdlib.h>
#include <stdio.h>

int main() {

  for(int n = 5; n > 0; n--) {
    printf("%d\n", n);
  }

  return 0;
}
```

```c
/* add2.c */
#include <stdlib.h>
#include <stdio.h>

int main() {

  int in;
  int result = scanf("%d", &in);

  while(result != EOF) {
    printf("%d\n", in+2);
    result = scanf("%d", &in);
  }

  return 0;
}
```

You have a Linux computer and all possible programs. How would you in the simplest possible way make the output from the first program, `ones`, be read by the the other program `add2`.

Part 2. Question 6

6 from one to the ...[P/F]

Assume that we have two programs, `ones` and `add2`, implemented as bellow. The call to `scanf("\%d", \&in)` will read from `stdin` and parse a number that is then stored in `\&in`. The procedure either returns 1, if it manages to read number, or `EOF`. The call to `printf()` will write the number to `stdout`.

```c
/* ones.c */
#include <stdlib.h>
#include <stdio.h>

int main() {

  for(int n = 5; n > 0; n--) {
    printf("%d\n", n);
  }

  return 0;
}
```

```
$> ./ones | ./add2
7
6
5
4
3
```

```c
/* add2.c */
#include <stdlib.h>
#include <stdio.h>

int main() {

  int in;
  int result = scanf("%d", &in);

  while(result != EOF) {
    printf("%d\n", in+2);
    result = scanf("%d", &in);
  }

  return 0;
}
```

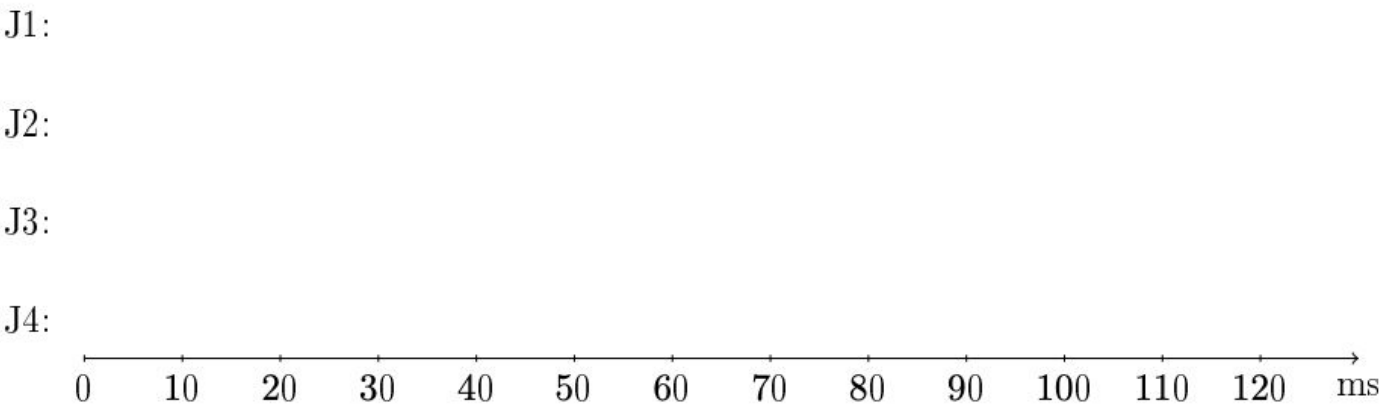
You have a Linux computer and all possible programs. How would you in the simplest possible way make the output from the first program, `ones`, be read by the the other program `add2`.

Part 2. Question 7

7 Scheduling [P/F]

Assume that we have a scheduler that implements *shortest job first*. We have four jobs described below as ⟨*arrive at*, *execution time*⟩ in ms. Draw a time diagram and specify the turnaround time for each of the jobs.

- J1 : ⟨0,40⟩
- J2 : ⟨0,30⟩
- J3 : ⟨10,10⟩
- J4 : ⟨20,30⟩



Part 2. Question 7

turnaround = finish time - arrival time

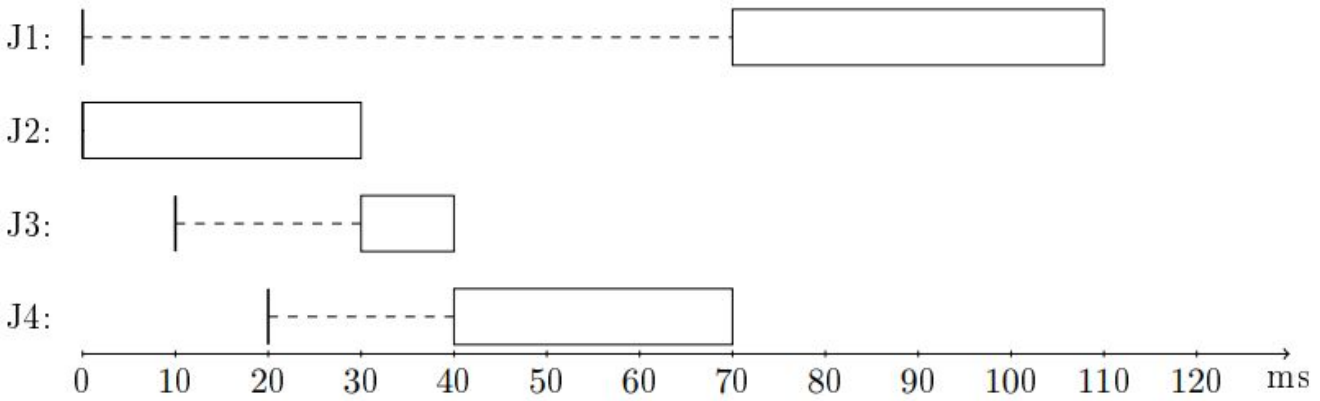
- J1 = 110 - 0 = 110 ms
- J2 = 30 - 0 = 30 ms
- J3 = 40 - 10 = 30 ms
- J4 = 70 - 20 = 50 ms

7 Scheduling [P/F]

Assume that we have a scheduler that implements *shortest job first*. We have four jobs described below as ⟨*arrive at*, *execution time*⟩ in ms. Draw a time diagram and specify the turnaround time for each of the jobs.

Answer:

- J1 : ⟨0,40⟩ 110 ms
- J2 : ⟨0,30⟩ 30 ms
- J3 : ⟨10,10⟩ 30 ms
- J4 : ⟨20,30⟩ 50 ms



Part 2. Question 8

8 you win some,you loose some [P/F]

Assume that we have a paged virtual memory with a page size of 4Ki byte. Assume that each process has four segments (for example: code, data, stack, extra) and that these can be of arbitrary but given size. How much will the operating system loose in internal fragmentation?

Part 2. Question 8

8 you win some,you loose some [P/F]

Assume that we have a paged virtual memory with a page size of 4Ki byte. Assume that each process has four segments (for example: code, data, stack, extra) and that these can be of arbitrary but given size. How much will the operating system loose in internal fragmentation?

Answer: Each segment will in average give rise to 2Ki byte of fragmentation. This will in average mean 8 Ki byte per process.

If we for example have 100 processes this is a total loss of 800 Ki byte.

Part 2. Question 9

9 paged memory with 64 byte pages [P/F]

You have been asked to propose an architecture for a processor that should have a paged virtual memory with the page size as small as 64 byte. The processor is a 16 bit processor and the virtual address space should be 2^{16} bytes.

Propose a scheme that uses a hierarchical page table based on pages of 64 Ki byte and explain how the address translation is done.

Part 2. Question 9

9 paged memory with 64 byte pages [P/F]

You have been asked to propose an architecture for a processor that should have a paged virtual memory with the page size as small as 64 byte. The processor is a 16 bit processor and the virtual address space should be 2^{16} bytes.

Propose a scheme that uses a hierarchical page table based on pages of 64 Ki byte and explain how the address translation is done.

Answer: One proposal is to use an offset of 6 bits and then have two levels in the tree with an index of 5 bits on each level. We need 6 bits as offset to address 64 bytes. The 5 bit used as index would mean that we could have 32 elements in a page table. If we encode each entry as two bytes we can encode a table in a page of 64 bytes. Using two bytes as an entry should be sufficient given that we only need 5 bits for an index, we have 11 bits for flags etc.

Part 2. Question 10

10 log-based fs [P/F]

In a log-based file system we write all changes in a continuous log without changing the existing data blocks that has been allocated to a file. We will sooner or later run out of blocks and need to reclaim blocks that are no longer used.

How do we keep track of which blocks that can be reused and what do we do to reclaim the blocks?

Part 2. Question 10

10 log-based fs [P/F]

In a log-based file system we write all changes in a continuous log without changing the existing data blocks that has been allocated to a file. We will sooner or later run out of blocks and need to reclaim blocks that are no longer used.

How do we keep track of which blocks that can be reused and what do we do to reclaim the blocks?

Answer: We need to identify the used block in the very back of the log. If we can move this block to the front of the log we can reused all consecutive blocks up to the next used block that is now the last used block in the log.

We maintain an inverse mapping that for a given block will tell us the inode of that the block belongs to. This means that we can determine if a block is used an if so, to which inode it belongs. If we copy the last block in the log to the front we also make a new copy of its inode with an updated sequence of blocks.

Unix Commands

ls – list files and directories

mkdir – create directory

rmdir – remove directory

cd – change directory

pwd – path to current directory

touch – create a file

rm – remove files or directories

mv – move a file

cp – copy files or directories

ln – create a link to a file

chmod – change permissions of a file

cat – print file to standard output

echo – display a line of text

head – output the first part of file

tail – output the last part of file

diff – compare files line by line

sort – sort lines of text files

wc – newline, word and byte count for file

sed – stream editor

grep – find a pattern in a file

tr – translate or delete characters

man – manual

Question 11

5.1 parking lots [2 points]

When they arranged for parking space along Sveavägen (central Stockholm) there were two alternatives: 1/ have painted parking lots of 6m in length or 2/ let cars park with 25 cm distance without the limitation of painted lots. If we, for simplicity, assume that cars are between 4.0 and 5.5 meters and that everyone can park a car in a slot with half a meter of extra space, then what is the problem with each of the solutions?

Question 11

5.1 parking lots [2 points]

When they arranged for parking space along Sveavägen (central Stockholm) there were two alternatives: 1/ have painted parking lots of 6m in length or 2/ let cars park with 25 cm distance without the limitation of painted lots. If we, for simplicity, assume that cars are between 4.0 and 5.5 meters and that everyone can park a car in a slot with half a meter of extra space, then what is the problem with each of the solutions?

Answer: In the first alternative we will have internal fragmentation since we loose in average 75 cm in each lot. The second alternative will risk having external fragmentation since empty spaces can be two small for most cars.

Question 12

1.2 memory map [2 points]

Below is a, somewhat shortened, printout of a memory mapping of a running process. Briefly describe the role of each segment marked with ???.

```
> cat /proc/13896/maps

00400000-00401000 r-xp 00000000 08:01 1723260            .../gurka ???
00600000-00601000 r--p 00000000 08:01 1723260            .../gurka ???
00601000-00602000 rw-p 00001000 08:01 1723260            .../gurka ???
022fa000-0231b000 rw-p 00000000 00:00 0                  [???]
7f6683423000-7f66835e2000 r-xp 00000000 08:01 3149003    .../libc-2.23.so ???
    :
7ffd60600000-7ffd60621000 rw-p 00000000 00:00 0          [???]
7ffd60648000-7ffd6064a000 r--p 00000000 00:00 0          [vvar]
7ffd6064a000-7ffd6064c000 r-xp 00000000 00:00 0          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0  [vsyscall]
```

Question 12

1.2 memory map [2 points]

Below is a, somewhat shortened, printout of a memory mapping of a running process. Briefly describe the role of each segment marked with ???.

```
> cat /proc/13896/maps

00400000-00401000 r-xp 00000000 08:01 1723260            .../gurka ???
00600000-00601000 r--p 00000000 08:01 1723260            .../gurka ???
00601000-00602000 rw-p 00001000 08:01 1723260            .../gurka ???
022fa000-0231b000 rw-p 00000000 00:00 0                  [???]
7f6683423000-7f66835e2000 r-xp 00000000 08:01 3149003    .../libc-2.23.so ???
    :
7ffd60600000-7ffd60621000 rw-p 00000000 00:00 0          [???]
7ffd60648000-7ffd6064a000 r--p 00000000 00:00 0          [vvar]
7ffd6064a000-7ffd6064c000 r-xp 00000000 00:00 0          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0  [vsyscall]
```

Answer: The first three segments are: code, read-only data and global data for the running process `gurka`. Then there is a segment for the *heap*. The segment marked with `lib-2.23.so` is a shard library. In the uppermost region we find the segment of the *stack*.

Question 13

2.1 count [2 points]

What will be printed if we execute the procedure `hello()` below concurrently in two threads? Motivate your answer.

```
int loop = 10;

void *hello() {
  int count = 0;

  for(int i = 0; i < loop; i++) {
      count++;
  }
  printf("the count is %d\n", count);
}
```

Question 13

2.1 count [2 points]

What will be printed if we execute the procedure `hello()` below concurrently in two threads? Motivate your answer.

```
int loop = 10;

void *hello() {
  int count = 0;

  for(int i = 0; i < loop; i++) {
      count++;
  }
  printf("the count is %d\n", count);
}
```

Each thread has its own version of **count** and the two threads will not disturb each other. Therefore, each thread will print “the count is 10”

Question 14

5.1 what could happen [2 points]

Assume that we have simple file system without a journal where we write directly to bitmaps, inodes and data data blocks. Assume that we shall write to a file and that an additional data block is needed. When we perform the operations on disc, we only succeed in updating the inode but not the bit maps nor the selected data block before we crash.

If we do not detect the error when we restart, which problems will we have and what could happen?

Question 14

5.1 what could happen [2 points]

Assume that we have simple file system without a journal where we write directly to bitmaps, inodes and data data blocks. Assume that we shall write to a file and that an additional data block is needed. When we perform the operations on disc, we only succeed in updating the inode but not the bit maps nor the selected data block before we crash.

If we do not detect the error when we restart, which problems will we have and what could happen?

Answer: We will have a data block that is allocated to an inode but the data block contains garbage and it is marked as free in the bit-maps. If we read from the file we will read garbage but worse if we use the data block for another file. This new file will then write its data to the block that can then be over written when we write to the first file. If the data block is used to represent a directory, this could of course result in total chaos.

Question 15

1.1 stack or heap [2 points]

What is done in the procedure below and where should `gurka` be allocated? Why? Complete the code so that `gurka` is allocated space.

```
void tomat(int *a, int *b) {
  // allocate room for gurka


  gurka = *a;
  *a = *b;
  *b = gurka;
}
```

Question 15

1.1 stack or heap [2 points]

What is done in the procedure below and where should `gurka` be allocated? Why? Complete the code so that `gurka` is allocated space.

```
void tomat(int *a, int *b) {
  // allocate room for gurka
```

int gurka;

```
  gurka = *a;
  *a = *b;
  *b = gurka;
}
```

Answer: The procedure switches value of the two arguments. The variable should be allocated on the stack since it has a known size and is not needed after the call to *tomat()*. This is done by adding a local declaration `int gurka;`.

Question 16

4.1 aproximate what? [2 points]

The clock algorithm, that is used to swap pages from memory, is described as an approximation. What is it that it tries to approximate? Describe a scenario when it does not do the right choice because of the approximation.

Question 16

4.1 aproximate what? [2 points]

The clock algorithm, that is used to swap pages from memory, is described as an approximation. What is it that it tries to approximate? Describe a scenario when it does not do the right choice because of the approximation.

Answer: It approximates LRU (*least recently used*). It only notes that a page has been used since the last turn, not how often or when it was accessed. If it is time to swap a page and all pages are marked as being used, the algorithm will clear the markers one by one and then swap the first page. It could be that this page was the page that was access most recently and thus should not be swapped.

Question 17

2.2 pipes [2 points]

The program below opens a pipe and iterates a number of times (ITERATIONS) where each iteration sends a number (BURST) of messages ("0123456789"). We need to handle the situation where the receiving process will not keep up with the sender; how do we implement flow-control to avoid buffer overflow?

```
int main() {
  int mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
  mkfifo("sesame", mode);
  // add flow control


  int flag = O_WRONLY;
  int pipe = open("sesame", flag);

  /* produce quickly */
  for(int i = 0; i < ITERATIONS; i++) {
    for(int j = 0; j < BURST; j++) {
      write(pipe, "0123456789", 10);
      // add flow control


    }
    printf("producer burst %d done\n", i);
  }
  printf("producer done\n");
}
```

2.2 pipes [2 points]

If we have two processes, one producer and one consumer, that are communicating through a so called *pipe*. How can we then prevent that the producer sends more information than the consumer is ready to receive and thereby crash the system.

Question 17

2.2 pipes [2 points]

The program below opens a pipe and iterates a number of times (ITERATIONS) where each iteration sends a number (BURST) of messages ("0123456789"). We need to handle the situation where the receiving process will not keep up with the sender; how do we implement flow-control to avoid buffer overflow?

```
int main() {
  int mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
  mkfifo("sesame", mode);
  // add flow control


  int flag = O_WRONLY;
  int pipe = open("sesame", flag);

  /* produce quickly */
  for(int i = 0; i < ITERATIONS; i++) {
    for(int j = 0; j < BURST; j++) {
      write(pipe, "0123456789", 10);
      // add flow control


    }
    printf("producer burst %d done\n", i);
  }
  printf("producer done\n");
}
```

Answer: Pipes have built-in flow controll. If the consumer does not read from the pipe the producer will be suspended when it tries to write the filled pipe.

2.2 pipes [2 points]

If we have two processes, one producer and one consumer, that are communicating through a so called *pipe*. How can we then prevent that the producer sends more information than the consumer is ready to receive and thereby crash the system.

Question 18

1.1 fork() [2p]

When you create a process using `fork()` the two processes will **share** some structures. Which, if any, of the following will the two processes share.

- Stack
- Heap
- Global memmory
- Code area
- Open files

Question 18

1.1 fork() [2p]

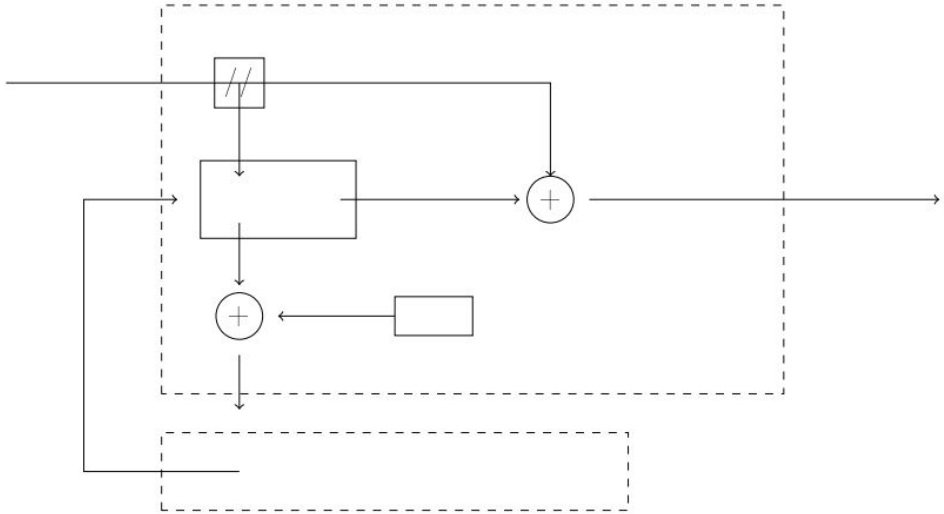
When you create a process using `fork()` the two processes will **share** some structures. Which, if any, of the following will the two processes share.

- Stack **Answer:** no
- Heap **Answer:** no
- Global memmory **Answer:** no
- Code area **Answer:** no
- Open files **Answer:** yes

Question 19

4.1 A paging MMU with TLB [2p]

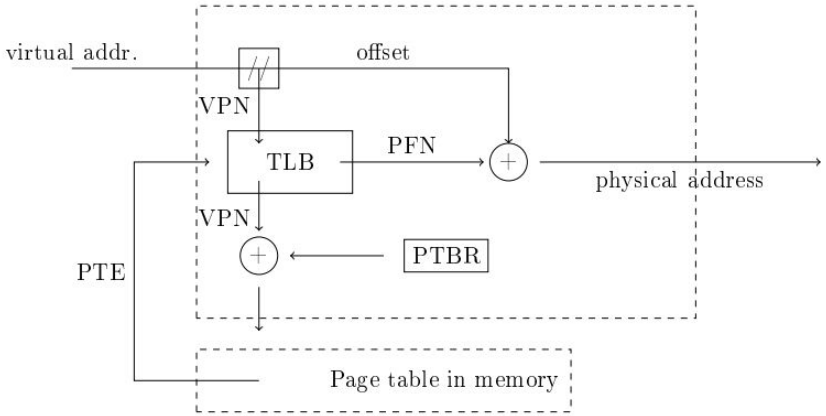
Below is a picture of a MMU that uses a TLB to translate virtual addresses to physical addresses. Identify the following units and addresses: virtual address, physical address, page table base register (PTBR), offset in page, page number (VPN), frmae number (PFN), TLB, page table, page table entry (PTE).

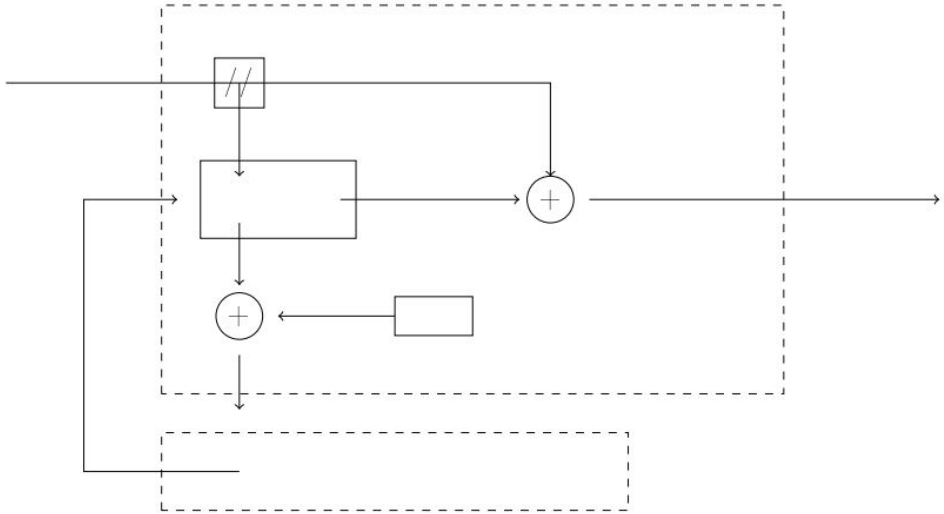


Question 19

4.1 A paging MMU with TLB [2p]

Below is a picture of a MMU that uses a TLB to translate virtual addresses to physical addresses. Identify the following units and addresses: virtual address, physical address, page table base register (PTBR), offset in page, page number (VPN), frmae number (PFN), TLB, page table, page table entry (PTE).





Question 20

3.2 reaction time [2 points]

When we want to reduce the reaction time we want to preempt a job even though the job is not completed. If we choose to do this we have one parameter to set, by changing this we can improve the reaction time. Which parameter is it? How should it be set and what unwanted consequence might it have?

Question 20

3.2 reaction time [2 points]

When we want to reduce the reaction time we want to preempt a job even though the job is not completed. If we choose to do this we have one parameter to set, by changing this we can improve the reaction time. Which parameter is it? How should it be set and what unwanted consequence might it have?

Answer: We can decrease the time slot given to a process. Doing so will decrease the reaction time of processes. Jobs that are ready to run will be scheduled much quicker. The disadvantage is that we will increase the turnaround time and in the worst case spend a large part of the time switching between processes.

Questions?