2022-a - Spørgsmål til Eksamen F22 Operativsystemer 02159

Operativsystemer (Danmarks Tekniske Universitet)

Technical University of Denmark

Written Examination, December 9, 2022

Course: Operating Systems

Course No.: 02159

Aids allowed: All aids

Exam duration: 4 hours

Weighting: According to their respective points (100 points in total)

Notes:

- All questions contribute toward the final grade. The maximum number of points awarded by each correctly answered question is listed next to the question.
- For short questions, the following apply: (i) a brief answer is expected; (ii) you are also expected to briefly explain your answer in approximately 2-3 lines; (iii) a correct answer with a correct explanation yields 4 points in total.
- For long questions, the following apply: (i) there might be more than one correct answer; (ii) you are expected to briefly explain your answer in approximately half a page; (iii) a complete and correct answer yields 12 points; (iv) if you make additional assumptions, remember to briefly explain them.

A. Short Questions

Question A.1: Name four (4) methods that can be used for communication between two processes that run on the same computer? Explain briefly your answer. (**4 points**)

Question A.2: Assuming the operating system uses priority-based scheduling, I/O bound processes should be treated as high or low priority? Explain briefly your answer. (**4 points**)

Question A.3: What will the code in Figure 1 print? Explain briefly your answer. You can assume that all system call invocations are successful. (**4 points**)

Question A.4: What will the code in Figure 2 print? Explain briefly your answer. You can assume that all system call invocations are successful. (**4 points**)

Question A.5: What will the code in Figure 3 print? Explain briefly your answer. You can assume that all system call invocations are successful. (**4 points**)

Question A.6: What will the code in Figure 4 print? If the code is executed multiple times, will the printed numbers always be in the same order? Explain briefly your answer. You can assume that all system call invocations are successful. (**4 points**)

Question A.7: You are developing an application that is composed of multiple threads. Each thread updates a global variable. Which method you would use to avoid a race condition? Explain briefly your answer. (**4 points**)

Question A.8: Assuming the code in Figure 5 is executed and the first print statement prints 20465, which terminal command will make the program terminate with exit code 7? Explain briefly your answer. You can assume that all system call invocations are successful. (**4 points**)

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int *counter;

void child(void) {
    ++*counter;
    exit(0);
}

int main(void)
{
  counter = (int*)malloc(sizeof(int));
  *counter = 1;

  if (fork() == 0) {
    child();
  }
  waitpid(-1, NULL, 0);

  printf("%d\n", *counter);
  return 0;
}
```

Figure 1: Code of Question A.3.

```
#include <stdio.h>
#include <pthread.h>

int A = 0;
pthread_mutex_t m;
pthread_t thread_id[100000];

void* count(void *input) {
  pthread_mutex_lock(&m);
  A++;
  pthread_mutex_unlock(&m);
  pthread_exit(NULL);
}

int main(void) {
  int i;
  pthread_mutex_init(&m, NULL);
  for (i=0;i<100000;i++)
    pthread_create(&thread_id[i], NULL, count, NULL);
  for (i=0;i<100000;i++)
    pthread_join(thread_id[i], NULL);
  printf("%d\n", A);
  return 0;
}
```

Figure 2: Code of Question A.4.

```
#include <pthread.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

#define SIZE 4096
char buf1[SIZE] = "hello\0";
char buf2[SIZE] = "world\0";
pthread_mutex_t lock1, lock2;
pthread_t tid1, tid2;

struct thread_args {
  char *a, *b;
  pthread_mutex_t *lock_a, *lock_b;
};

struct thread_args t1 = {.a = buf1, .b = buf2,
  .lock_a = &lock1, .lock_b = &lock2};
struct thread_args t2 = {.a = buf2, .b = buf1,
  .lock_a = &lock2, .lock_b = &lock1};

void *swap(void *args) {
  FILE *temp;

  pthread_mutex_lock(((struct thread_args*)args)->lock_a);
  temp = tmpfile();
  fprintf(temp, "%s", ((struct thread_args*)args)->a);

  pthread_mutex_lock(((struct thread_args*)args)->lock_b);
  memcpy(((struct thread_args*)args)->a,
    ((struct thread_args*)args)->b, SIZE);
  rewind(temp);
  fscanf(temp, "%s", ((struct thread_args*)args)->b);
  fclose(temp);

  pthread_mutex_unlock(((struct thread_args*)args)->lock_b);
  pthread_mutex_unlock(((struct thread_args*)args)->lock_a);
  pthread_exit(0);
}

int main(void) {
  pthread_mutex_init(&lock1, NULL);
  pthread_mutex_init(&lock2, NULL);

  pthread_create(&tid1, NULL, swap, (void*)&t1);
  pthread_create(&tid2, NULL, swap, (void*)&t2);
  pthread_join(tid1, NULL);
  pthread_join(tid2, NULL);
  printf("%s␣%s\n", buf1, buf2);
}
```

Figure 3: Code of Question A.5.

```c
#include <stdio.h>
#include <pthread.h>

pthread_t thread_id[5];

void* count(void *a) {
  printf("%d\n", *(int*)a);
  pthread_exit(NULL);
}

int main(void) {
  int i;
  for (i=1;i<=5;i++) {
    pthread_create(&thread_id[i], NULL, count, &i);
    pthread_join(thread_id[i], NULL);
  }
  return 0;
}
```

Figure 4: Code of Question A.6.

```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

int flag = 1;
void handler(int sig) {
  flag = 0;
}

int main(void){
  signal(SIGTERM, handler);
  printf("%d\n", getpid());
  while (flag)
    sleep(1);
  return 7;
}
```

Figure 5: Code of Question A.8.

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(void){
  int a = 0;
  fork();
  fork();
  fork();
  printf("%d\n", ++a);
  exit(0);
}
```

Figure 6: Code of Question A.9.

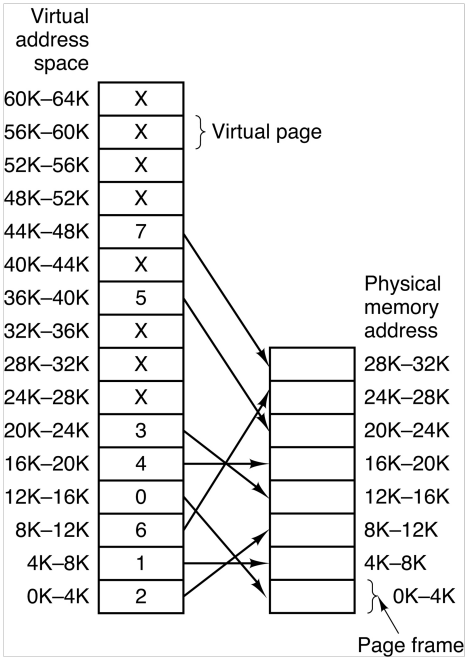

Figure 7: Data for Question A.12.

Question A.9: What will the code in Figure 6 print? Explain briefly your answer. You can assume that all system call invocations are successful. (**4 points**)

Question A.10: When implementing mutual exclusion with a spinlock is good for efficiency? Explain briefly your answer. (**4 points**)

Question A.11: Which is the key disadvantage of the FIFO (First-In, First-Out) page replacement algorithm? Explain briefly your answer. (**4 points**)

Question A.12: Assuming the current state of the memory is as shown in Figure 7, name a virtual address that will not generate a page fault if accessed? Explain briefly your answer. (**4 points**)

Question A.13: Name a scenario that it is efficient to use DMA for memory transfer? Explain briefly your answer. (**4 points**)

Question A.14: Name a system operation that is not possible without clock interrupts. Explain briefly your answer. (**4 points**)

Question A.15: Describe a scenario where interrupt-based software is resource efficient. Justify briefly your answer. (**4 points**)

Question A.16: What is the main advantage of multiprocessor systems compared to single-processor systems? Explain briefly your answer. (**4 points**)

B. Long Questions

Question B.1 (12 points)

(i) Summarise the advantages and disadvantages of processes and threads.

(ii) You are developing a Domain Name System (DNS) server. The purpose of a DNS server is to translate domain names (e.g. `www.dtu.dk`) to IP addresses (e.g. `192.38.84.35`). When a DNS server receives a request from a client, it first looks if this domain name is in a local cache, otherwise it makes a request to a higher-tier DNS server. Once it retrieves the IP address, it responds to the client. The DNS server should handle multiple requests from potentially hundreds of clients in parallel. Would you implement the DNS server using multiple processes (e.g. one process per client) or multiple threads (e.g. one thread per client)? Motivate your answer and discuss if the disadvantages of your solution are relevant in this use case. If relevant, also discuss how you would overcome them.

Question B.2 (12 points)

(i) Summarise the advantages and disadvantages of preemptive and nonpreemptive scheduling.

(ii) Elaborate on if, in the OS Challenge, it would be better to process the requests using preemptive or nonpreemptive scheduling?

Question B.3 (12 points)

(i) Let's assume that you are designing a server for the OS Challenge. Arriving requests have 95% probability to have priority 1 and 5% probability to have priority 250. How would you design parallelism and prioritisation? You shall assume that anything not explicitly specified in this sub-question is as described on the OS Challenge specification document.

(ii) Let's assume that you are designing a server for the OS Challenge. You expect to receive an unlimited number of requests, but your memory has room for a cache of only 100 entries. What would be the most efficient cache replacement policy? You shall assume that anything not explicitly specified in this sub-question is as described on the OS Challenge specification document.