1. Un proces execută la un moment dat:

```
sigaction(SIGUSR1, &sa, NULL);
```

iar la un moment ulterior

```
write(fd, buffer, 4);
```

În care din situații este mai probabilă înlocuirea majorității intrărilor din TLB? Motivați. (Argumentele și modul de folosire a funcțiilor se prespun corecte.)

Problema se tratează cel mai bine de la coadă la cap. Care sunt situațiile în care se înlocuiesc majoritatea intrărilor din TLB (eventual un flush – golire)? Răspuns: în cazul unei schimbări de context. Se schimbă tabelele de pagini între procesul preemptat și cel planificat, mai puțin partea kernel. TLB-ul se golește (dacă arhitectura și/sau sistemul de operare permite) atunci unele intrări rămân active (zone de memorie partajată comune procesulelor, zone din spațiul kernel). De aici cuvântul "majorității".

Când se realizează un context switch?

- *la terminarea unui proces*
- *la expirarea cuantei de rulare a unui proces*
- *în momentul în care prioritatea procesului curent (cel ce rulează) este sub prioritatea unui proces READY*
- *în cazul blocării procesului curent*

În cazul celor două apeluri doar ultima variantă are sens (blocarea procesului curent). Acest lucru se poate întâmpla doar în cazul apelului write, care este un apel blocant.

2. Care este limita de spațiu de swap pentru un sistem cu magistrala de adrese de 32 de biți cu spațiul de adrese împărțit 2GB/2GB (user/kernel). Dar pentru un sistem cu magistrala de adrese de 64 de biți?

Nu există nici o limitare. Singura limitare este cea impusă de hardware.

3. O funcție signal-safe este o funcție care poate fi apelată fără restricții dintr-o rutină de tratare a unui semnal (signal handler). De ce nu este malloc o funcție signal-safe? Oferiți o secvență de cod pentru exemplificare.

După cum s-a menționat în cateva rezolvări (fără a aduce o contribuție în cadrul răspunsului, însă) funcțiile signal-safe sunt practic echivalente cu funcțiile reentrante. O funcție non-signal-safe este o funcție care folosește variabile statice, astfel că, dacă un semnal întrerupe funcția în programul principal și funcția este rulată în handler este posibil să apară inconsistențe (exact cum se întâmplă în momentul în care un thread este întrerupt și rulează alt thread fără asigurarea accesului exclusiv și consistent la date).

Dacă un semnal întrerupe funcția malloc și, în handler, rulează funcția malloc structurile interne folosite de libc pentru gestiunea alocării memoriei procesului vor fi date peste cap. Funcția printf este, în mod similar, o funcție non-signal-safe pentru că folosește buffer-ele interne ale libc. Mai multe informații aici (https://www.securecoding.cert.org/confluence/x/34At)

Scenariul de exemplificare este de forma:

```
void sig_handler(int signo)
{
	void *p = malloc(100);
}

int main(void)
{
		....
	void *a = malloc(BUFSIZ);    /* aici sosește semnalul */
		...
}
```

Răspunsul "malloc poate genera SIGSEGV când deja rulează un signal handler" nu este valid. Malloc nu generează SIGSEGV; cel mult rămâne fără memorie și întoarce NULL. SIGSEGV este generat în momentul accesării unei regiuni invalide a memoriei.

4. Un program execută următoarea secvență de cod:

```
for (i = 0; i < BUFLEN; i++)
    printf("%c", buf[i]);
```

iar altul

```
for (i = 0; i < BUFLEN; i++)
    write(1, buf+i, 1);
```

Care secvență durează mai mult? De ce?

Funcția printf folosește buffering-ul din libc. Acest lucru înseamnă că, până la îndeplinirea uneia dintre cele trei acțiuni de mai jos, nu se face apel de sistem:

- *se umplu buffer-ele*
- *se apelează fflush(stdout)*

- *se transmite newline (\n)*

Apelul de sistem write face apel de sistem de fiecare dată rezultând un overhead important.

Pentru convingere puteți rula testul de aici (http://anaconda.cs.pub.ro/~razvan/school/so/test_printf_write.c). Mai jos este un exemplu de rulare, primul folosind printf al doilea folosind write (se alterează macro-ul USE_PRINTF). Rezultatele sunt, în opinia mea, edificatoare.

```
razvan@valhalla:~/school/2008-2009/so/examen$ time ./test_printf_write > out.txt
real   0m0.076s
user   0m0.060s
sys    0m0.020s

razvan@valhalla:~/school/2008-2009/so/examen$ time ./test_printf_write > out.txt
real   0m5.930s
user   0m0.052s
sys    0m5.868s
```

5. Un proces P se găsește în starea READY. Precizați și motivați două acțiuni care determină trecerea acestuia in starea RUNNING.

Fie Q procesul care rulează în acest moment pe procesor. Situații de trecere a lui P din READY în RUNNING:
- *Q se încheie și P este primul din coada de procese READY*
- *lui P îi este crescută prioritatea peste a lui Q*
- *lui Q îi expiră cuanta și P este primul în coada de procese READY*
- *Q efecuează o operație blocantă (trece în blocking) și P este primul proces în coada de procese READY*

6. De ce obținerea ordonată/ierarhică a lock-urilor previne apariția deadlock-urilor, respectiv apariția fenomenului de starvation?

Ordonarea modului de obținere (achiziție) a lock-urilor în particular și a resurselor în general previne apariția de cicluri în graful de alocare a resurselor și deci apariția deadlock-urilor.

În absența ordinii de obținere un proces P1 poate face Lock(1) și apoi Lock(2). Înainte de Lock(2) este preemptat și procesul P2 face Lock(2) și apoi încearcă Lock(1). Ambele procese rămân blocate în așteptare mutuală (deadly embrace) = deadlock.

Nu există nici o legătură directă în lock-uri și fenomenul de starvation. Fenomentul de starvation caracterizează o durată de așteptare foarte mare pentru un proces gata de rulare. Alte procese îi iau tot timpul "fața" și procesul nu ajunge pe procesor. Se spune că sistemul nu este "fair" (echitabil). Principala formă de asigurare a echității este folosirea noțiunii de cuantă și, în lumea Linux, de epocă și folosirea priorității dinamice a proceselor. Orice formă de locking duce la creșterea nivelului de starvation. Un proces care așteaptă la un semafor intrarea într-o regiune critică dar alte procese intră înaintea sa. Asigurarea fairness-ului poate fi asigurată prin strategii de tipul FIFO. Dar, obținerea ierarhică a lock-urilor nu are un efect vizibil diferit față de folosirea în orice fel a lock-urilor din perspectiva starvation.