1. Care din următoarele instrucțiuni **ar putea** suprascrie adresa de retur a unei funcții? (my_func este o funcție) Motivați și precizați contextul în care se poate întâmpla. (Poate fi un singur răspuns, răspunsuri multiple, nici unul, toate răspunsurile)

```
long *a = malloc(30);  /* definire si alocare */

/* instructiuni */
a)  a = my_func;
b)  *(&a + 4) = my_func;
c)  *(a + 0x4000000) = my_func;
d)  memcpy(my_func, a, sizeof(void *));
```

Cuvânt cheie: ar putea. Unele situații sunt improbabile dar posibile.

Înainte de toate:
- *a este o variabilă (de tip pointer)*
- *a rezidă pe stivă*
- *&a este adresa pe stivă a variabilei a*
- *a (valoarea a) este o adresă de heap (punctează către zona de 30 de octeți alocată folosind malloc)*
- *în general, heap-ul crește în sus și stiva crește în jos*
- *my_func este o funcție deci rezidă în .text (zona de cod)*

a) nu are un efect; se suprascrie valoarea lui a cu adresa funcției my_func
*b) posibil; dacă există unele variabile între [ret][ebp] și [a] atunci &a+4 poate puncta către adresa unde se găsește valoarea de retur și *(&a + 4) o poate suprascrie*
c) posibil; a+0x4000000 înseamnă adunarea a 0x4000000 la o adresă de heap (valoarea lui a); se poate ajunge (greu probabil, dar posibil) la o adresă de retur de pe stivă
d) ciudată, se suprascrie o informație din zona de cod (zona read only); pot apărea erori de acces sau comportanent nedeterminist (în nici un caz nu suprascrierea adresei de retur dintr-o funcție)

2. Un proces este folosit pentru calcularea de transformate Fourier iar un altul este folosit pentru căutarea de informații într-o ierarhie de fișiere. Care dintre cele două procese va avea prioritatea mai mare? De ce?

Proces care calculează transformate Fourier – CPU intensive. Proces care caută informatii într-o ierarhie de fisiere – I/O intensive. În general, procesele I/O intensive au prioritate mai mare. Motivele sunt:
- *creșterea interactivității*
- *împiedicarea starvation (fairness); dacă nu ar fi astfel prioritizate, procesele CPU intensive s-ar transforma în "processor hogs" și ar folosi resursele sistemului*
- *procesele I/O intensive ocupă timp puțin pe procesor deci întârzierea provocată altor procese este mică*

3. Un sistem dispune de un TLB cu 128 de intrări; care este capacitatea maximă a memoriei fizice și a memoriei virtuale pe acel sistem?

Nu există nici o legătură. TLB-ul menține mapări de pagini fizice și pagini virtuale. Conține un subset al tabelei de pagini. Memoria fizică și memoria virtuală pot fi oricât de mici/mari. Nu sunt afectate de dimensiunea TLB-ului.

4. Completați zona punctată de mai jos cu (pseudo)cod Linux (POSIX) sau Windows (WIN32) (la alegere) care va conduce la afișarea mesajului "alfa" la ieșirea standard (standard output) și mesajul "beta" la ieșirea de eroare standard (standard error):

```
/* de completat */
[...]
fputs("alfa", stderr);
fputs("beta", stdout);
```

Nu alterați simbolurile standard fputs (functie), stderr și stdout (FILE *).

Problema este, de fapt, o problemă a paharelor ascunsă. Se dorește ca ieșirea standard să folosească descriptorul 2, iar ieșirea de eroare standard să folosească descriptorul 1.

În pseudocod Linux, lucurile stau astfel:

```
int aux_fd;

/* aux_fd punctează către ieșirea standard */
dup2(STDOUT_FILENO, aux_fd);
/* descriptorul de ieșire standard este închis și apoi punctează către ieșirea de eroare standard */
dup2(STDERR_FILENO, STDOUT_FILENO);
/* descriptorul de ieșire de eroare standard este închis și apoi punctează către ieșirea standard (indicată de aux_fd) */
dup2(aux_fd, STDERR_FILENO)

fputs ....
```

5. Fie următoarea secvență de (pseudo)cod:

```
int *a;
a = mmap(NULL, 4100, PROT_READ | PROT_WRITE, MAP_ANONYMOUS | MAP_PRIVATE, -1, 0);
n = read_int_from_user();
a[n] = 42;
n = read_int_from_user();
a[n] = 42;
```

Ce efect au valorile introduse de utilizator asupra programului? (page faults, erori, scrieri în memorie) Discuție. (o pagină ocupă 4 KB; read_int_from user() citește o valoare întregă de la intrarea standard)

Discuția este ceva mai amplă. Prerequisites:

- *mmap lucrează la nivel de pagină*
- *mmap nu alocă memorie fizică "din prima"; se alocă la acces (demand paging) în urma unui page fault*

Nu am considerat necesar să se observe că a este int și că referirea primei pagini se face cu 0 <= n <= 1024. Au fost considerată validă și observația 0 <= n <= 4096 pentru prima pagină.*

Se solicită alocarea a 4100 de octeți (> 4096, < 8192) deci se vor aloca 2 pagini.

Primul read:

- *n < 0, proababil eroare (SIGSEGV) în cazul în care pagina anterioară este nevalidă (destul de probabil)*
- *n >= 2048 (peste cele două pagini), probabl eroare (SIGSEGV)*
- *0 <= n < 1024 page fault și alocare spațiu fizic și validare pagină pentru prima pagină; fără erori*
- *1024 <= n < 2048 la fel ca mai sus pentru a doua pagină; fără erori (chiar și pentru n >= 1025 (4100/4))*

Al doilea read:

- *n < 0 idem*
- *n >= 2048 idem*
- *n este în aceeași pagină ca mai sus nu se întâmplă nimic*
- *n în cealaltă pagină atunci page fault, alocare spațiu fizic și validare pagină*

Practic mmap(..., 4100, ...) este echilvalent cu mmap(..., 8192, ...).

6. Care este avantajul configurării întreruperii de ceas la valoarea de 1ms? Dar la valoarea de 100ms?

1ms

- *timp de răspuns scurt, interactivitate sporită, fairness, sisteme desktop (întreruperi dese, se diminuează timpul de așteptare pentru fiecare proces*

100ms

- *productivitate (throughtput) sporită, sisteme server (mai puține context switch-uri, mai mult timp pentru "lucru efectiv")*