UHASSELT KU LEUVEN

EXAMENFORMULIER

Naam student(e):	Behaald cijfer: /
Jaar: ~~1B /~~ 2B ~~/ 3B / SCH / MA / GIT~~ groep:	Volgnummer: Studentennummer:
Academiejaar: 2020-2021	
Datum examen: 18/06/2021	

Naam/code opleidingsonderdeel	4082: Besturingssystemen en C (BESC)
Studiepunten	6
Naam docent(en)	Wouter Groeneveld & Robin Marx

Opleiding/Afstudeerrichting	IIW 2Ba Elektronica-ICT	
Examenvorm	☒ Schriftelijk ☐ Schriftelijk met mondelinge toelichting ☐ Mondeling met schriftelijke voorbereiding ☐ Permanente evaluatie	
	☒ 1ste examenkans	☐ 2de examenkans
Blok	Blok 1 ☐ 2 ☐ 3 ☐ 4 ☐	
Semester	Semester 1 ☐ 2 ☒	

Opmerkingen i.v.m. de organisatie van het examen

- Je krijgt tijd van **08h30 tot 11h30** om het examen af te leggen.
- Je mag géén gebruik maken van een computer en/of rekentoestel.
- Noteer onmiddellijk je **naam en volgnummer op elk deel**!
- De vragen worden op het examenpapier opgelost. Er is voldoende plaats.
- Kladpapier wordt niet verbeterd of bekeken!
- Het examen bestaat uit **7 vragen**

VEEL SUCCES!

Vraag 1: Definities

Leg volgende begrippen (kort) uit:

- External Fragmentation
- Atomic Operation
- Cooperative Scheduling
- Stack Canary

Vraag 2: Analyseer en corrigeer indien nodig

Analyseer de onderstaande stukjes C-code. Ontbrekende includes mag je negeren. Bepaal voor ieder stuk exact 1 soort gedrag:

1. Compileert niet
2. Compileert wel, maar werkt niet zoals verwacht
3. Compileert wel en werkt zoals verwacht

Indien optie 1 of 2 wordt gekozen, verbeter de fouten.

C-Code (+ eventuele verbetering)	Gedrag 1 / 2 / 3

`void maths(int* x) { x++; }`
`int main() {`
`  int one = 0;`
`  maths(&one);`
`  return one == 1; }`	

`struct dinges { void* x; void* y; };`
`int main() {`
`  int a = 3;`
`  struct dinges iets = { (void*) &a, (void*) &a };`
`  free(&a); printf("%d\n", iets.x);`
`}`	

`int main() {`
`  printf("%s", hello());`
`  return 0;`
`}`
`char* hello() { return "heykes"; }`	

`void s(int** x, int** y) {`
`  int *z = *x;`
`  **x = **y;`
`  **y = *z;`
`}`
`int* m(int i) {`
`  int *t = malloc(sizeof(int)); *t = i; return t; }`
`int main() {`
`  int *two = m(2), *one = m(1);`
`  *two = 2;`
`  *one = 1;`
`  s(&one, &two);`
`}`	

Vraag 3: Multiple choice

- Duid het juiste antwoord aan. Er is *exact* 1 juist antwoord per vraag.
    - correct antwoord: +1
    - foutief antwoord: -1/3
    - blanco: 0

Dit is *niet* één van de drie minimale functionaliteiten van een OS kernel:	○ Task management ○ Inter-process communication ○ Device drivers ○ Address space management
Threads in een multi-threaded process hebben een:	○ individuele stack ○ individuele heap ○ individueel text/code segment ○ individuele lijst van open files
Een multi-level feedback queue scheduler:	○ heeft altijd 3 levels ○ is altijd beter voor interactieve taken ○ heeft nooit last van starvation ○ garandeert dat een taak altijd slechts in 1 run queue tegelijk zit

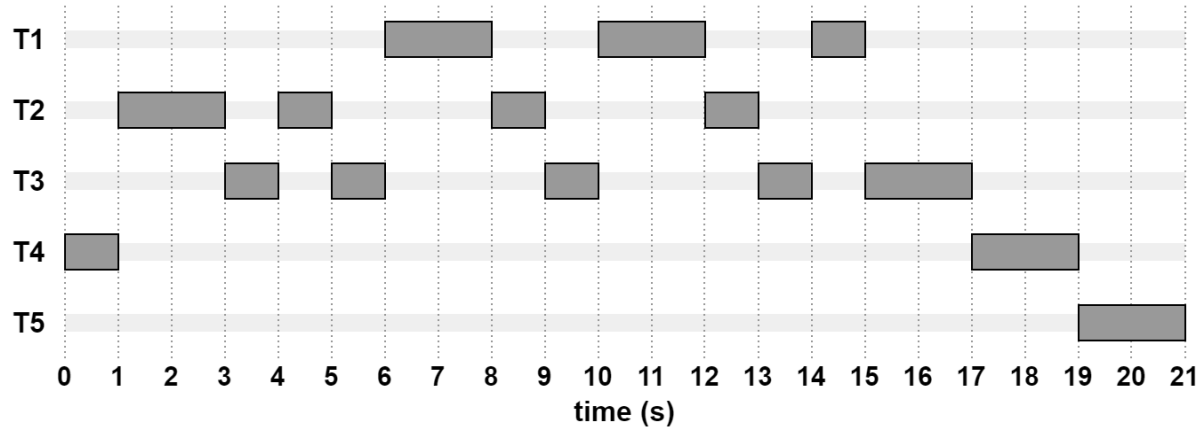
Hoe vaak wordt de letter 'r' geprint volgens onderstaande code:
`pid_t pid = fork();`
`if (pid != 0) { // parent`
`  pid = fork();`
`}`
`if (pid != 0 ) { // parent`
`  fork();`
`}`
`printf("r\n");`
`fork();`
	○ 3
	○ 4
	○ 5
	○ 6
	○ 7
	○ 8
	○ 9
	○ 10

Gegeven een 6-bit processor met pages/frames van 8 bytes. Deze gebruikt een 2-level hierarchical page table waarbij de outer page table 2 elementen bevat. Het virtuele geheugenadres is daarbij opgebouwd uit volgend aantal bits voor de inner page table (p2) en voor de offset (d):	○ p2: 2 bits, d: 3 bits ○ p2: 3 bits, d: 2 bits ○ p2: 1 bit, d: 3 bits ○ p2: 2 bits, d: 4 bits ○ p2: 3 bits, d: 3 bits
Geheugenbeheer technieken hebben vaak belangrijke nadelen. Een voorbeeld van zulk een *nadeel* is:	○ chunking geeft interne fragmentatie ○ paging vraagt (veel) extra geheugen ○ segmentatie zorgt voor meer segmentation faults

Vraag 4: Scheduling

A) Gegeven onderstaand scheduling schema van een **priority-based** scheduler:

1) Bepaal de (*geschatte*) aankomsttijd, duur en prioriteit van elk van de 5 taken *(ga ervan uit dat taak 5 aankomt op 10 seconden)*
2) Bepaal de context switching overhead
3) Bereken AJCT
4) Is dit een voorbeeld van een SJF scheduler? Waarom wel/niet?
5) Zijn er verschillen te merken in hoe de individuele prioriteitsniveaus taken schedulen? Zo ja: leg uit welke. Zo nee: leg uit waarom niet.



B) Het concept **prioriteit** kan op verschillende manieren geïnterpreteerd, geïmplementeerd en geïntegreerd worden in een scheduling algoritme. Bespreek 2 opties en geef voor- en nadelen van beide.

Vraag 5: Code

1. Gegeven een syllabus die bestaat uit verschillende bladzijden. Beeld je in dat je door de syllabus aan het bladeren bent: je kan van bladzijde 3 naar bladzijde 4 bladeren (vooruit), maar ook van bladzijde 4 naar bladzijde 3 (achteruit). Veralgemeend wordt gesteld: elke bladzijde refereert naar de vorige en volgende bladzijde.

   A) Werk een struct uit in C uit die deze datastructuur voorstelt.
   B) Hoe ga je om met de eerste en laatste bladzijden?

2. **A)** Wat is een segmentation fault?

   B) Waarom geeft de volgende code in C een segmentation fault?

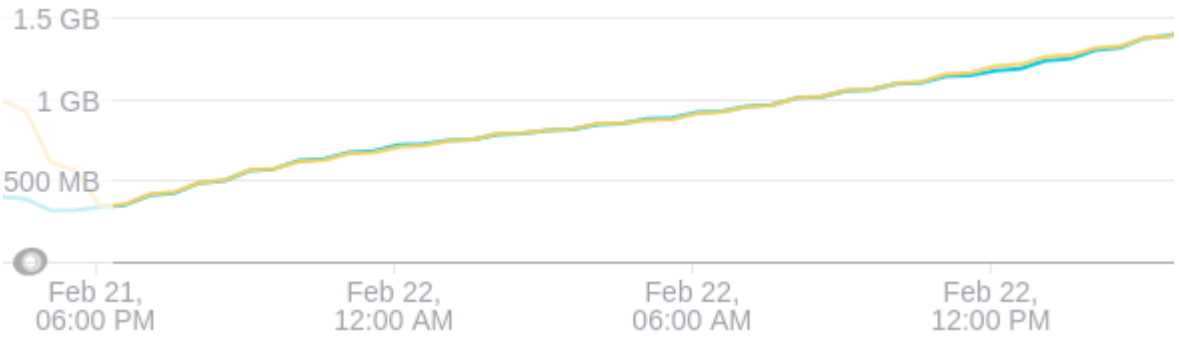
```
int hoeveel = 0; scanf("%d", hoeveel);
printf("echt zoveel? %d\n", hoeveel);
```

   C) Geef nog **twee andere** situaties in C waarin een segfault kan voorkomen.

Vraag 6: geheugenbeheer

Gegeven onderstaande grafiek die het geheugengebruik visualiseert van een applicatie:

1. Beschrijf en situeer wat je ziet. Welk fenomeen is duidelijk zichtbaar?
2. Wat stel je voor om hier aan te doen? Teken hoe de grafiek er zou uitzien als we je voorstellen hebben geïmplementeerd.
3. Gaat het hier over de stack of de heap? Waarom?



Vraag 7: Multithreading

Bekijk onderstaand deel van een multithreaded producer/consumer implementatie:

```
01. int shelf;
02.
03. void producer(int from, int to) {
04.   for (int i = from; i < to; ++i) {
05.     shelf = i;
06.     sem_post(&produced);
07.     sem_wait(&consumed);
08.   }
09. }
10. void consumer() {
11.   while( true ) {
12.     sem_wait(&produced);
13.     printf("%d was produced \n", shelf );
14.     sem_post(&consumed);
15.   }
16. }
```

1. Beschrijf aan de hand van een concreet stappenplan (met thread- en lijnnummers) hoe er een race condition kan optreden in deze code als er meerdere producers en consumers *tegelijk* actief zijn. Geef aan waar de kritische sectie(s) zich bevind(t)(en). Beschrijf ook de mogelijke fout die hieruit voortvloeit (toon daarbij voorbeeld output).
2. Herschrijf de nodige delen van de code om deze fout op te lossen zodat het programma werkt met meerdere gelijktijdige producer en consumers. Geef daarbij ook de nodige (pseudo)code om de semaforen correct te initialiseren.
3. Momenteel gaat deze code niet correct stoppen, want de consumer blijft altijd in de while(true). Geef twee concrete manieren (met kort voorbeeld) waarop je dit soort probleem veilig kan stoppen (dus zonder pthread_kill, en enkel als *alle* producers gedaan zijn en *alle* data verwerkt is door de consumers).