Klausur Betriebssysteme und Sicherheit, 23.07.2024

— Bearbeitungszeit: 90 Minuten — Prüfer: Prof. Dr. Schirmeier, Prof. Dr. Tschorsch —

1	2	3	4	5	6	$\sum$
15	16	15	15	15	14	90

Alle Aussagen sind so ausführlich wie nötig, aber so knapp wie möglich zu begründen.

Aufgabe 1 – Sicherheitslücken

15 Punkte

Eine Sicherheitsforscherin entdeckt und veröffentlicht Details zu einer Schwachstelle sofort nach ihrer Entdeckung. Im weiteren Verlauf stellt sich heraus, dass die Sicherheitslücke nach der Veröffentlichung ausgenutzt wird.

a) Handelt sich dabei um Full Disclosure oder Responsible Disclosure? Begründen Sie Ihre Antwort in wenigen Worten.

2 P

b) Ein Exploit benutzt nur diese Schwachstelle. Handelt es sich dabei um einen 0-day Exploit oder 1-day Exploit? Begründen Sie Ihre Antwort in wenigen Worten.

2 P

Im Folgenden finden Sie Eigenschaften zur Verbreitung oder Funktionsweise von Malware. Klassifizieren Sie basierend auf dieser Beschreibung die Malware.

c) Nach der Infektion beginnt die Software Dateien auf dem Computer des Opfers zu verschlüsseln und verlangt eine Zahlung (normalerweise in Bitcoin) im Austausch für den Schlüssel der Verschlüsselung. **1 P**

d) Die Software tarnt sich als wünschenswerter Code oder Software, wie z. B. eine Banking-App oder ein Spiel, um in ein System einzudringen. **1 P**

e) Die Software kann sich selbst über ein Netzwerk übertragen, um andere Computer zu infizieren sowie sich selbst vervielfältigen. **1 P**

f) Die Software infiziert andere Programme und reproduziert sich damit selbst. **1 P**

Die Funktion `secret_check` vergleicht eine Eingabe (`input`) mit dem Passwort vom System (`secret`). Hierbei lädt die Funktion `load_secret` das Passwort vom System. Die Funktion `strcmp` vergleicht zwei Strings und gibt `0` zurück, wenn die beiden Strings gleich sind.

Abbildung 2 zeigt den Stack unmittelbar vor dem `return` (Zeile 11) am Ende der Funktion `secret_check`.

```
1  void load_secret(char*);
2  void unlock(void);
3  void backdoor(void);
4
5  void secret_check() {
6    char secret[8];
7    char input[8];
8    load_secret(secret);
9    gets(input);
10   if (strcmp(input, secret) == 0)
11     unlock();
12   return;
13 }
```

Abbildung 1: C-Code.

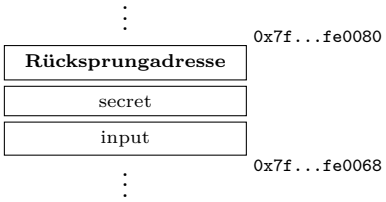


Abbildung 2: Stack.

g) Warum stellt ein Buffer Overflow ein Sicherheitsrisiko dar? **1 P**

h) Konstruieren Sie eine mögliche Eingabe um den Programmablauf zu modifizieren und die Funktion `backdoor` auszuführen. Nehmen Sie an, dass die Funktion `backdoor` an der Adresse `0x0000000000000351` beginnt. Gehen Sie dabei auf die Bestandteile der Eingabe ein und warum diese notwendig sind. Sie dürfen die Byte-Reihenfolge (Endianness) selbst bestimmen.

4 P

i) Nennen und erklären Sie zwei Schutzmechanismen, die gegen Buffer Overflows wirken.

2 P

Aufgabe 2 – Kryptografie **16 Punkte**

Abbildung 3 skizziert einen TLS-Handshake.

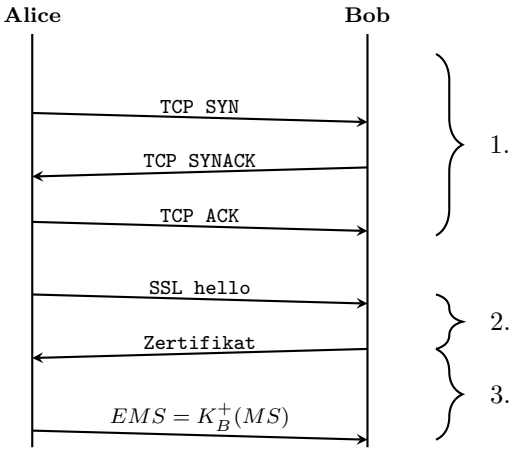


Abbildung 3: Vereinfachter TLS Handshake.

a) Beschreiben Sie den Ablauf der einzelnen Schritte 1 bis 3 in wenigen Worten. **3 P**

b) Welche Schutzziele werden nach einem erfolgreichen TLS-Handshake für die Kommunikation zwischen Client und Server gewährt?

3 P

c) Beschreiben Sie das Angreifermodell für einen erfolgreichen Monster-in-the-Middle (MitM) Angriff gegen einen TLS-Handshake. Welches Wissen muss ein Angreifer besitzen bzw. was muss ein Angreifer können, um den Angriff erfolgreich durchzuführen?

Nehmen Sie hierbei an, dass die kryptografischen Grundvoraussetzungen bestehen, insbesondere können die kryptografischen Primitiven selbst nicht gebrochen werden.

2 P

d) Könnten wir prinzipiell im Rahmen des TLS-Handshakes auch den Diffie-Hellman (DH) Schlüsselaustausch verwenden? Falls ja, wo müsste der DH Schlüsselaustausch hinzugefügt werden? Geben Sie in diesem Fall den entsprechenden Schritt an. Falls nein, begründen Sie Ihre Antwort in wenigen Worten.

2 P

Gehen Sie von einer 2-Bit Blockchiffre mit der in Tabelle 1 dargestellten Schlüsseltabelle aus.

Tabelle 1: Schlüsseltabelle

In	Out
00	01
01	11
10	00
11	10

e) Verschlüsseln Sie den Klartext `01010000` mit der Betriebsart CBC (Cipher Block Chaining). Der Initialisierungsvektor ist 00.

3 P

f) Nennen Sie zwei Unterschiede von CBC gegenüber Electronic Codebook (ECB).

2 P

g) Welches Problem haben beide Verfahren (CBC und EBC)?

1 P

Aufgabe 3 – Scheduling

15 Punkte

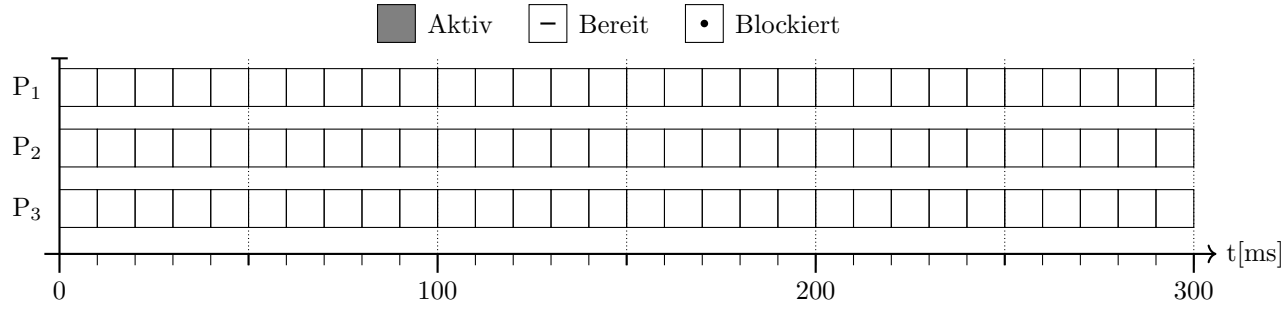
Ein Einprozessor-Betriebssystem verwaltet drei Prozesse P_1, P_2 und P_3. Die Prozesse treffen in dieser Reihenfolge im System ein und sind alle zum Zeitpunkt $t = 0$ rechenbereit. Die Prozesse wiederholen sich unendlich lange. Nach jedem CPU-Stoß führen die Prozesse einen E/A-Stoß durch. Die CPU-Stöße (in ms) und E/A-Stöße (in ms) der Prozesse sind in der Tabelle 2 angegeben.

Prozesse	P_1	P_2	P_3
CPU-Stöße	30	40	60
E/A-Stöße	50	40	30

Tabelle 2: Prozesslaufzeiten

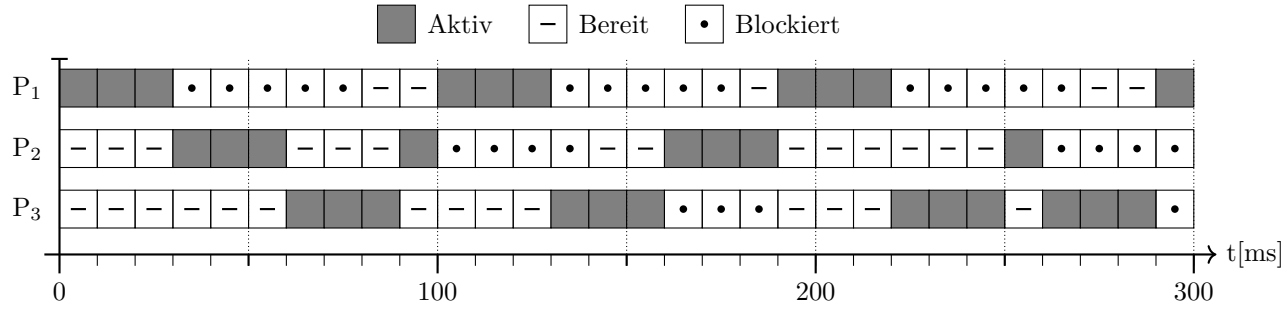
a) Zeichnen Sie in das folgende Gantt-Diagramm ein, wie die drei Prozesse P_1, P_2 und P_3 bearbeitet werden würden, wenn das Scheduling nach der „Shortest Remaining Time First“-Strategie vorgenommen wird. Nehmen Sie an, dass die Prozesse die in der Tabelle 2 gegebenen Laufzeiten haben und der Scheduler auf Basis dieses Wissens entscheiden kann.

Falls Sie ihre Antwort korrigieren möchten, können Sie die Ersatzdiagramme auf der letzten Seite nutzen. **Streichen Sie in diesem Fall ungültige Lösungen deutlich durch!**



6 P

b) Nehmen Sie wieder an, dass die Prozesse die in der Tabelle 2 gegebenen Laufzeiten haben. Das folgende Gantt-Diagramm zeigt eine mögliche Abarbeitungsreihenfolge der Prozesse. Welche Scheduling-Strategie aus der Vorlesung führt zu diesem Verhalten? Bitte kreuzen Sie eine Strategie an. Erklären Sie zudem kurz an zwei eindeutigen Merkmalen, warum Sie sich für diese Strategie entschieden haben.
Die Zahl in Klammern bei „Round Robin“ und „Virtual Round Robin“ ist jeweils die Zeitscheibenlänge.



- ☐ Shortest Process Next
- ☐ Virtual Round Robin (30 ms)
- ☐ Round Robin (30 ms)
- ☐ First-Come First-Served

3 P

c) Welches Problem von „Round Robin“ (RR) wird beim „Virtual Round Robin“ (VRR) vermieden?

1 P

In einem Einprozessor-Echtzeitsystem sind drei periodische Tasks T_1, T_2 und T_3 einzuplanen mit folgenden Werten (p_i Periodenlänge, t_i Bearbeitungszeit, Periodenende = Zeitschranke):

$p_1 = 6$, $t_1 = 1$ $\quad$ $p_2 = 10$, $t_2 = 3$ $\quad$ $p_3 = 5$, $t_3 = 1$

Zwischen den Tasks bestehen keine Abhängigkeiten und sie sind an beliebiger Stelle unterbrechbar.

d) Ist die Taskmenge einplanbar mittels statischer Prioritäten? Falls ja, schlagen Sie einen Algorithmus vor und geben Sie die Zuordnung der Prioritäten an. Ansonsten begründen Sie Ihre Aussage.

3 P

e) Die Taskmenge aus Teilaufgabe (d) soll nun mit dynamischen Prioritäten eingeplant werden. Jedoch entsteht durch die dynamische Einplanung ein Scheduling-Overhead. Dieser verlängert die Bearbeitungszeit der Tasks um jeweils eine Zeiteinheit. Begründen Sie, ob diese modifizierte Taskmenge mit dynamischen Prioritäten auf einem Einprozessorsystem einplanbar ist.

2 P

Aufgabe 4 – Speicherverwaltung und Seitenersetzung **15 Punkte**

In einem 52-Bit System wird eine 3-stufige Seitentabelle zur Verwaltung der 2^{52} Byte großen virtuellen Adressräume verwendet. Jede Stufe ist gleich groß und hat dabei 4096 Einträge. Eine Seitentabelle ist so groß wie eine Seite. In diesem System wächst der Stack abwärts.

a) Beantworten Sie folgende Fragen:

1. Wie viele Bits werden für die Adressierung jeweils einer Stufe benötigt?

2. Wie viele Bits der virtuellen Adresse sind Offset?

3. Wie groß ist eine Kachel?

4. Wie groß ist ein Seitentabelleneintrag?

4 P

Gegeben sei weiterhin folgende Tabelle der verwendeten Speicherbereiche eines Programms:

Bereich	**Beginn**	**Ende**	**Schutzbits**
Code	`0x10000`	`0x5FFFF`	r,x
Data	`0x60000`	`0x7FFFF`	r,w
Heap	`0xA0000`	`0xCFFFF`	r,w
Stack	`0xB00D0000`	`0xBFFFFFFF`	r,w

Außerdem verwendet das Betriebssystem folgende Seitentabellen für die Verwaltung des Speicherbereiches des Programms (nicht angegebenen Bereiche der Seitentabellen sind nicht *present*) und hat noch folgende freien Kacheln (`0xC0FFE0000`, `0xBEEF0000`, `0xBABE0000`). Die Seitentabelle der ersten Stufe liegt bei `0x20000`.

0x20000		
`0x0`	`1`	p, w, x

0x10000		
`0x0`	`4`	p, w, x
`0x1`	`4`	w, x
...	...	...
`0xB`	`4`	p, w

0x40000		
`0x0`	`800`	w, x
`0x1`	`AFFE`	p, x
`0x2`	`DEAD`	p, x
`0x3`	`F00`	w
`0x4`	`AFFE`	w, x
`0x5`	`DEAD`	w
`0x6`	`F00`	p, w
...	...	...
`0xC`	`1234`	p, w
`0xD`	`800`	p, w
...	...	...
`0xFFF`	`ABC`	p, w

b) Das Programm führt die folgenden Zugriffe durch. Protokollieren Sie die – wenn möglich – daraus resultierende physische Adresse, sowie alle Zwischenschritte. Sollte eine Übersetzung nicht möglich sein, notieren Sie, wo die Übersetzung fehlschlägt.

1. Lesender Zugriff auf `0x1ABCD`

2. Schreibender Zugriff auf `0xADEAD`

3. Schreibender Zugriff auf `0xB00DF006`

9 P

c) Die Adresse des aktuell verwendeten Stackframes liegt bei `0xB00D0005`. Jetzt wird die unten stehende Funktion aufgerufen. Was passiert bei dem Zugriff auf die Variable `mydata` in Zeile 4? Nennen Sie mögliche Ergebnisse dieses Zugriffes und eventuelle Reaktionen des Betriebssystems.

```
1 void foobar(int x) {
2     int mydata[100];
3     for (int i = 0; i < 100; i++)
4         mydata[i] = x;
5 }
```

2 P

Aufgabe 5 – Unix

15 Punkte

a) Nennen Sie jeweils das Segment, in dem die Symbole `counter`, `hallo`, `main` und `fd0` gespeichert werden.

`counter`	
`hallo`	
`main`	
`fd0`	

2 P

Bedeutung der genutzten Flags:

- `O_CREAT`: Wenn die angegebene Datei nicht existiert, wird eine neue Datei erzeugt.
- `O_WRONLY`: Die angegebene Datei wird ausschließlich zum Schreiben geöffnet.

```
int counter;
char *hallo = "Hallo Welt";
char *test  = "Test";

int main(void) {
    int fd0 = open("a.txt",
     ↪ O_CREAT|O_WRONLY);
    counter++;
    int pid = fork();
    int fd1 = open("b.txt",
     ↪ O_CREAT|O_WRONLY);

    if(pid == 0) {
        write(fd0, hallo, 10);
        write(fd1, hallo, 10);
    } else {
        write(fd0, test, 4);
        wait();
        write(fd1, test, 4);
        counter++;
    }

    printf("Counter: %d", counter);
    return 0;
}
```

b) Welchen Inhalt hat Datei `a.txt` nach der erfolgreichen Ausführung des Programms? Falls mehrere Dateiinhalte möglich sind, nennen Sie diese.

2 P

c) Welchen Inhalt hat Datei `b.txt` nach der erfolgreichen Ausführung des Programms? Falls mehrere Dateiinhalte möglich sind, nennen Sie diese. Begründen Sie Ihre Aussage.

2 P

d) Welche Ausgabe steht nach der erfolgreichen Ausführung des Programms in der Konsole?

2 P

e) Um das gegebene C-Programm ausführen zu können, muss der Quellcode in Maschinencode übersetzt werden. Wählen Sie die dazu nötigen Schritte aus der unten stehenden Liste aus und notieren Sie die zugeordneten Ziffern in der korrekten Reihenfolge.

Hinweis: Die Auswahl falscher Schritte führt zu Punktabzug innerhalb dieser Teilaufgabe!

1. Kompilieren
2. Dateien a.txt und b.txt im Dateisystem anlegen
3. Assemblercode in Objektdatei umsetzen
4. Gegebenen C-Code durch Präprozessor verarbeiten
5. Maschinencode in den Kernel laden
6. Stack des gegebenen C-Programms aufbauen
7. Bibliotheken dazubinden
8. Data-Segment des gegebenen C-Programms in den Hauptspeicher laden
9. Bankalgorithmus ausführen

5 P

f) Die ausführbare Datei `program` liest während der Ausführung die Datei `data` ein. Setzen Sie die Zugriffsrechte für `program` so, dass die beiden Nutzer `adam` und `eva` die Datei `program` erfolgreich ausführen können und das Programm die Datei `data` erfolgreich einlesen kann. Die Zugehörigkeit von Benutzern zu Gruppen darf nicht geändert werden.

Dateiname	Besitzer	Gruppe	Rechte
`program`	`root`	`root`	
`data`	`root`	`root`	`rw- rw- ---`

2 P

Aufgabe 6 – Synchronisation

14 Punkte

Der Zugriff auf eine Datenstruktur soll mit Hilfe eines Reader-Writer-Lock geschützt werden. Dabei erhält entweder ein Schreiber exklusiven, schreibenden Zugriff auf die Datenstruktur (`write_exclusive`) oder mehrere Leser dürfen parallel die Datenstruktur lesen (`read_parallel`).
Falls Sie ihre Antwort korrigieren möchten, können Sie die Ersatzdiagramme auf der letzten Seite nutzen. **Streichen Sie in diesem Fall ungültige Lösungen deutlich durch!**

a) Vervollständigen Sie die gegebene Funktionen `write_exclusive` und `read_parallel` und verwenden Sie hierfür die Semaphore `res_sem` und `count_sem`. Initialisieren Sie diese mit den richtigen Startwerten. Verwenden Sie die Zählvariable `count`, um die aktuelle Anzahl der Leser zu verfolgen. Sie dürfen und müssen nur Quellcode an den markierten Stellen einfügen, aber es müssen nicht alle Lücken gefüllt werden. Achten Sie besonders darauf, richtig zu synchronisieren, um Synchronisationsprobleme wie Deadlocks oder Race Conditions zu vermeiden. **8 P**

b) Obwohl beliebig viele Leser parallel die Datenstruktur lesen können, sollen aus Ressourcengründen maximal 5 Leser gleichzeitig lesen dürfen. Erweitern Sie die Funktion `read_parallel_limited` mit Hilfe der Semaphore `queue_sem` und initialisieren Sie diese mit dem richtigen Wert. **3 P**

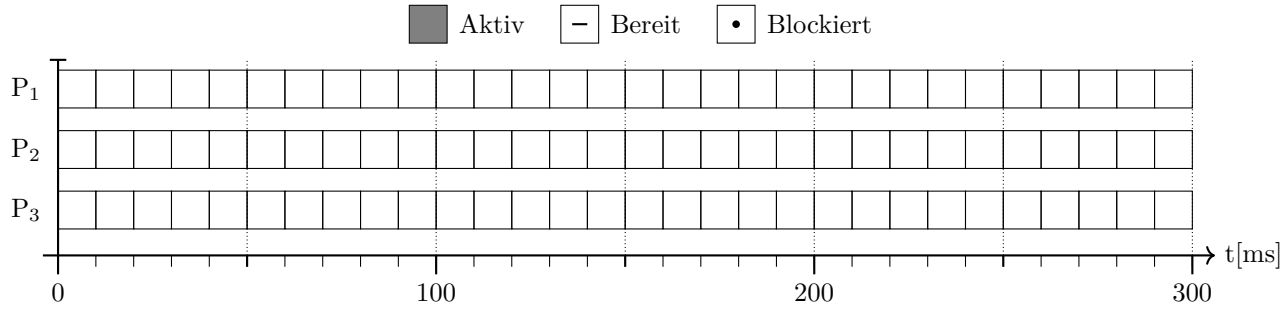
```
sem_t queue_sem = sem_t(    )

item read_parallel_limited() {
    
    item it = read_parallel();
    
    return it;
}
```

```
sem_t count_sem = sem_t(    )
sem_t res_sem   = sem_t(    )
int count       = 0

void write_exclusive(item& it) {
    
    write_data(it);
    

}

item read_parallel() {
    
    if(count == 0) {
        
    }
    
    count = count + 1;
    
    item it = read_data();
    
    count = count - 1;
    
    if(count == 0) {
        
    }
    

    return it;
}
```

c) Beschreiben Sie das Problem des *Aushungerns* (Starvation). Warum kann es in der Implementierung einer Reader-Writer-Datenstruktur dazu kommen? **3 P**

Ersatzdiagramme

Für Aufgabe 3 b)



Für Aufgabe 6 a)

```
sem_t count_sem = sem_t(      )
sem_t res_sem   = sem_t(      )
int count       = 0

void write_exclusive(item& it) {
     
     write_data(it);
     
}

item read_parallel() {
     
     if(count == 0) {
          
     }
     
     count = count + 1;
     
     item it = read_data();
     
     count = count - 1;
     
     if(count == 0) {
          
     }
     

     return it;
}
```

Für Aufgabe 6 c)

```
sem_t queue_sem = sem_t(      )

item read_parallel_limited() {
     
     item it = read_parallel();
     

     return it;
}
```