VO 182.711 6. Juni 2012

# Prüfung Betriebssysteme

KNr. MNr. Zuname, Vorname

| Ges.)(100) | 1.)(35) | 2.)(20) | 3.)(45) | Zusatzblätter: |
|---|---|---|---|---|
| | | | | |

**Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!**

# 1 Synchronisation mit Semaphoren (35)

Eine Parkgarage hat $N$ Stellplätze für Autos. An Ein- und Ausfahrt befinden sich je ein Schranken, um das Zu- bzw. Abfahren von Autos zu kontrollieren. Vor dem Schranken der Einfahrt befindet sich außerdem ein Ticket-Automat, bei dem vor dem Einfahren ein Ticket gelöst werden muss. An der Ausfahrt steht ein Automat, der die Bezahlung von Tickets überprüft, bevor der Ausfahrtschranken geöffnet wird.

Definieren bzw. initialisieren Sie geeigenete Semaphore und schreiben Sie drei Prozesse, *einfahrt*, *ausfahrt* und *auto*, um das Parken von Autos in der Parkgarage zu simulieren.

- *einfahrt* steuert den Ablauf an der Garageneinfahrt für beliebig viele Autos. Folgende Funktionen werden von diesem Prozess verwendet:
  *print_ticket()*: druckt ein Ticket für ein Auto aus.
  *open_gate(gate_no)*: öffnet den Schranken mit der als Parameter übergebenen Nummer (1 ... Einfahrt, 2 ... Ausfahrt).
  *close_gate(gate_no)*: schließt den Schranken, dessen Nummer als Parameter übergeben wurde (1 ... Einfahrt, 2 ... Ausfahrt).

- *ausfahrt* steuert den Ablauf an der Garagenausfahrt für beliebig viele Autos. Neben den Funktionen *open_gate(gate_no)* und *close_gate(gate_no)* wird folgende Funktion verwendet:
  *fetch_ticket()*: zieht ein Ticket ein, bevor der Schranken geöffnet wird. Nehmen Sie der Einfachheit halber an, dass alle Tickets bezahlt wurden und die Funktion immer erfolgreich terminiert.

- *auto* simuliert die Ankunft, das Parken und das Ausfahren eines Autos. Folgende Funktionen werden verwendet:
  *approach_gate(gate_no)*: Zufahren zum Schranken *gate_no* (Nummern wie oben beschrieben).
  *pass_gate(gate_no)*: Passieren des Schranken *gate_no*.
  *get_ticket()*: Nehmen eines an der Einfahrt gedruckten Tickets.
  *insert_ticket()*: Ticket wird in den Automaten an der Ausfahrt gesteckt.
  *park_and_pay()*: Parken incl. Bezahlen des Tickets.

Schreiben Sie Implementierungen für die drei Prozesstypen, die die beschriebenen Aktionen der Prozesse geeignet synchronisieren und sicherstellen, dass sich nie mehr als $N$ Autos in der Parkgarage befinden. Synchronisieren Sie die Prozesse ausschließlich mit Semaphoren. Die Verwendung von anderen Synchronisationskonstrukten und globalen Variablen ist nicht erlaubt.

Initialisierungen



/** Prozess einfahrt **/



/** Prozess ausfahrt **/

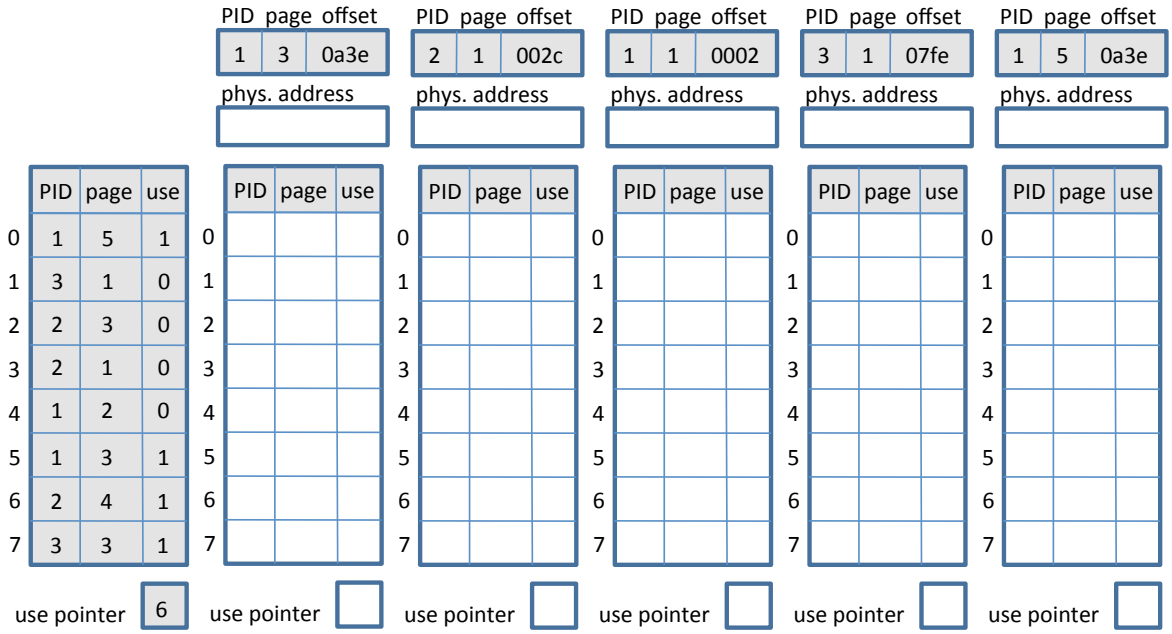


/** Prozess auto **/

# 2 Memory Management (20)

Die Page Table eines kleinen Computersystems ist als Inverted Page Table (IPT) mit acht Einträgen realisiert. Die Größe der einzelnen Pages beträgt 4kByte. Beim Auftreten von Page Faults wird in diesem Computersystem der Clock Algorithmus als globale Page Replacement Strategie eingesetzt.

Im zu lösenden Beispiel ist eine Ausgangsbelegung der Page Table (links), sowie eine Folge von Zugriffen auf den virtuellen Speicher, charakterisiert durch Prozessnummer (PID), adressierte Seite (page) und Offset, gegeben. Simulieren Sie die Ausführung der Zugriffsfolge von links nach rechts und tragen Sie für jeden Zugriff auf den virtuellen Speicher (a) die entsprechende physikalische Adresse, (b) den Zustand der IPT und (c) den Wert des Use Pointers für den Clock Algorithmus nach dem Speicherzugriff an (Sie brauchen bei jedem Schritt nur die Werte in der IPT anzugeben, die sich beim aktuellen Zugriff geändert haben).




Ändert sich für die konkrete Zugriffsfolge etwas an den Inhalten der IPT nach den einzelnen Schritten, wenn der Clock Algorithmus als lokale (anstatt globale) Ersetzungsstrategie angewandt wird? Wenn es zu einer Veränderung kommt, geben Sie diese bitte genau an.

# 3 Fragen zu Betriebssystemen (45)

Was versteht man unter einem *Process Control Block*? Beschreiben Sie, aus welchen Teilen der PCB besteht und welche Informationen in diesen Teilen jeweils verwaltet werden. (6)



Skizzieren Sie die Folge der Schritte, die bei der Abarbeitung eines *Synchronen I/O Requests* ablaufen. (5)

Was versteht man unter einem *Kernel Level Thread* und unter einem *User Level Thread*? Beschreiben Sie die beiden Arten der Threadimplementierung und charakterisieren Sie deren Unterschiede. (4)

Erklären Sie die Begriffe *Deadlock*, *Lifelock* und *Starvation*. (4)

Bei welchen der folgenden *Scheduling*-Strategien kann es zur *Starvation* kommen: (a) FCFS, (b) Shortest Job First, (c) Round Robin, (d) Priority Scheduling? Begründen Sie jeweils Ihre Antwort. (4)

Was versteht man unter *Blocking* bzw. *Non-Blocking I/O*? Beschreiben Sie die beiden Arten, I/O-Operationen durchzuführen. (3)

Was versteht man unter *Buffering*? Welche Vorteile bietet es, wo liegen seine Grenzen und worauf hat man bei der Verwendung von Puffern bei der Betriebssystemimplementierung zu achten? (5)

Nennen Sie vier verschiedene Strategien zur Block-Allokierung von Dateien und beschreiben Sie diese mit ihren Vor- und Nachteilen. (6)

Wie ist ein *i-node* aufgebaut? Welche Informationen enthält er? (4)

Nennen Sie Design Prinzipien für die Konstruktion von sicheren Systemen. Geben Sie für jede Regel ein Beispiel an. (4)