3. Signale, Interrupts, Scheduling

3.1 Signale

Es geht hier insgesamt darum, für typische Situationen, welche auftreten können, das Verhalten von Signalen auszutesten.

Aufgabe 1a: Signal-Handler

Schreiben Sie ein Programm in Python, in dem

1. Sie für verschiedene Signale einen Signal-Handler hinterlegen,
2. die Signal-Handler mit Zeitstempel auf der Console ausgeben, welches Signal sie empfangen haben.
3. das Hauptprogramm und/oder die Signal-Handler aus einer Schleife bestehen, in der entweder
   a) etwas gerechnet wird,
   b) ein blockierender System-Call erfolgt, oder
   c) mit sleep() für jeweils 10 Sekunden geschlafen wird.

Schicken Sie Ihrem Programm mit 'kill -s <signo>' die Signale SIGINT und SIGUSR1.

Aufgabe 1b: Signale

Fragen zu Signalen (als Anregung für Ihre Testsituationen):

1. was geschieht, wenn Sie Ihrem Programm
   a) ein Signal einmalig schicken, während es schläft / arbeitet / blockiert ist ?
   b) ein Signal mehrfach schicken, während es schläft / arbeitet / blockiert ist ?
   c) ein Signal ohne hinterlegten Handler schicken ?
2. welche Informationen können Sie in Signalen mit 'kill -s <signo>' an ihr Programm transportieren ?
3. können Sie Ihrem Programm auch von einem anderen Rechner aus ein Signal schicken ?
4. wie können Sie auch für Signale ohne explizit hinterlegten Signal-Handler folgende Reaktion erzeugen:
   a) Signal ignorieren ?
   b) ein Default-Verhalten, wenn sonst kein Handler hinterlegt ist ?

Letztlich sollen Sie alle Möglichkeiten und Situationen bei der Verwendung von Signalen abklären. Schauen Sie sich dazu u.a. auch die System-Calls signal(), sigsetmask(), sigblock() an.

3.2 Interrupts

Aufgabe 2: Interrupt-Behandlung

Beantworten Sie folgende Fragen stichwortartig:

1. Beschreiben Sie detailliert den Ablauf einer Interrupt-Bearbeitung.
   Hier kann man 10-12 Punkte benennen.

2. Wie wird die richtige Interrupt-Service-Routine bzw. deren Adresse ermittelt ?
3. Wie unterscheidet
   1. das BS,
   2. die CPU,

   ob diese sich in einer Interrupt-Bearbeitung befinden oder nicht ?

3.3 Scheduling

Aufgabe 3: Simulation von Schedulern

Dat Programm `sched.py` simuliert in der Funktion `schedule()` den FCFS-Scheduler auf Basis einer Prozesskonfiguration in der Datei `Datensatz.dat`.

Aufruf: `python sched.py <Datensatz.dat>`

Diese Dateien haben folgenden Aufbau:

```
1:1,6,3,4,-1
2:30,-1
4:4,1,4,1,4,-1
```

Dabei steht jede Zeile für einen Prozess. Am Anfang einer Zeile findet sich der Startzeitpunkt des Prozesses, dann folgen nach einem Doppelpunkt (durch Kommata getrennt) die Zeiten, in denen der Prozess CPU- und I/O-Phasen durchläuft – die erste Phase ist dabei eine CPU-Phase.

Ein Prozess, der mit I/O beginnt, müsste also in der Zeitenliste zunächst den Wert 0 (für eine CPU-Phase der Länge 0) enthalten. Außer direkt am Anfang darf in der Zeitenfolge keine 0 vorkommen. Die Abfolge ...,3,0,4,... müsste zu ...,7,... zusammengefasst werden. Der letzte Eintrag (-1) in jeder Zeile kennzeichnet das Prozessende.

1. Erzeugen Sie eine Konfigurationsdatei `nur-cpu.dat` für die vier Beispielprozesse auf Folie 4.Scheduling-29 und überprüfen Sie die Ausführreihenfolge.
2. Machen Sie sich mit dem Programmquelltext vertraut – er ist gut dokumentiert.
   Welche Aufgaben übernehmen die Funktionen
   a) `create_process()`,
   b) `run_current()` und
   c) `update_blocked_processes()` ?
3. Erzeugen Sie eine Kopie von `sched.py` (etwa: `sched-sjf.py`) und passen Sie darin die Funktion `schedule()` so an, dass sie statt FCFS den SJF-Scheduler (Shortest Job First) implementiert.

   Kommentieren Sie das Ergebnis stichwortartig.
4. Etwas komplizierter wird es, einen unterbrechenden Scheduler zu schreiben.
   a) Im letzten Schritt passen Sie eine Kopie von `sched-sjf.py`, z. B. `sched-srt.py`, an und implementieren nur den SRT-Scheduler (Shortest Remaining Time).
   Hier funktioniert die einfache Variante aus den vorigen zwei Schedulern nicht mehr, aktive Prozesse solange laufen zu lassen, bis sie sich beenden oder in eine I/O-Phase eintreten.

b) Kommentieren Sie Ihre Programmsourcen

c) Zeichnen Sie ein Gant-Diagramm Ihres Ergebnisses.
Prüfen Sie, dass Ihr Ergebnis mit dem erwarteten Ergebnis übereinstimmt !

Aufgabe 4: Scheduler Rechenaufgabe

Fünf Prozesse treffen zu gegebenen Zeitpunkten in der Bereit-/Ready-Liste ein. Es ist bekannt, wie viel Rechnenzeit sie benötigen. Jeder Prozess hat eine Priorität (0 stellt die höchste Priorität dar).

Die folgende Tabelle gibt die Ankunftszeiten, Rechenzeiten und Prioritäten der einzelnen Prozesse wieder:

Prozess	**Ankunftszeit**	**Rechenzeit**	**Priorität**	**Wartezeit**	**Ausführungs-dauer**
A	0	8	4		
B	3	28	1		
C	7	12	0		
D	9	3	2		
E	15	4	3		
Mittelwerte:					

Betrachten Sie die Ausführung der Prozesse unter verschiedenen Scheduler-Strategien.

1. Nicht-präemptiv (ohne Verdrängung), d.h. nach der Zuteilung der CPU arbeitet ein Prozess, bis er von selbst den Prozessor abgibt (oder blockiert).
   a) First Come First Served (FCFS)
   b) Prioritätsbasiertes Scheduling (Highest Priority First = HPF)
   c) Shortest Job First (SJF)
2. Prämptiv (mit Verdrängung) d.h. der Scheduler kann die CPU entziehen wenn anderer Prozess rechenbereit geworden ist oder nach Ablauf einer bestimmten Zeit.
   a) Round Robin (RR) mit dem Zeitscheibenwert (Zeitquantum) q = 5, ohne Prioritäten
   b) HPF
   c) Shortest Remaining Time First. Dies ist die prämptive Variante von SJF, die die CPU an den Prozess mit der kürzesten Restzeit zuweist.

Teilaufgaben für die gegebenen Scheduler-Strategien:

1. zeichen Sie die Gantt-Diagramme
2. berechnen Sie je Prozess die Werte in der Tabelle:
   a) die Wartezeit,
   b) die Ausführungsdauer (oder auch Laufzeit),
   c) die jeweiligen Mittelwerte über alle Prozesse.

Hier zur Selbstkontrolle die Mittelwerte:

1a) FCFS:	21,8 und 32,8	präemptiv:	2a) RR:	14,4 und 25,4
1b) HPF:	18,6 und 29,6		2b) HPF:	24,8 und 35,8
1c) SJF:	8,8 und 19,8		2c) SRT:	6,4 und 17,4