INTERPROZESSKOMMUNIKATION UND SCHEDULING

Hinweis: Wie in den vorangegangenen Aufgabenblättern, wird für die folgenden Übungen ein Linux-Betriebssystem vorausgesetzt. Lesen Sie bitte auch die zugehörige Praktikums-dokumentation durch.

1. PIPES

Die Intervallhalbierung oder auch Bisektion ist die Grundlage zur effizienten Lösung vieler Probleme in der Informatik. Im Nachfolgenden soll die Intervallhalbierung genutzt werden, um eine Zahl zu erraten.

Ein Prozess erzeugt einen Kindprozess. Der Kindprozess erzeugt eine Zufallszahl, welche im gesamten Bereich von `unsigned int` liegen kann. Der Vaterprozess fragt beim Kind-prozess solange an, ob seine aktuelle Zahl die richtige ist, bis er eine positive Antwort be-kommt.

Der Kindprozess antwortet immer folgendermaßen:

- -1, wenn die geratene Zahl kleiner als die Zufallszahl ist,
- 0, wenn die geratene Zahl die Zufallszahl ist,
- +1, wenn die geratene Zahl größer als die Zufallszahl ist.

Aufgrund der Informationen des Kindprozesses kann der Vaterprozess in jeder Iteration das Intervall, in dem die Zufallszahl liegen kann, halbieren.

Ein Programm, welches den geschilderten Sachverhalt abbildet, wurde Ihnen in Moodle zur Verfügung gestellt.

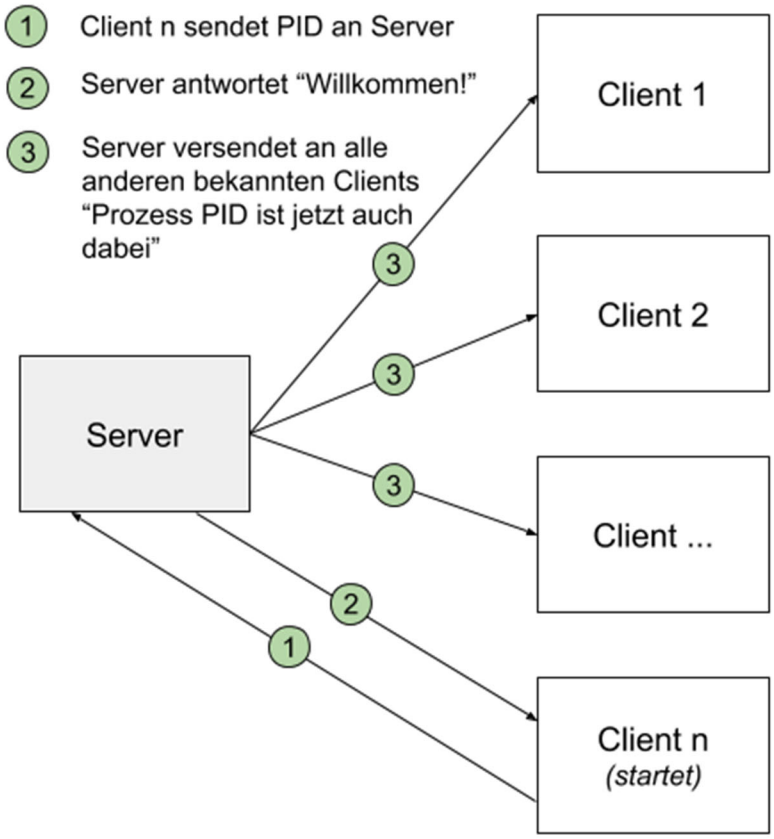
Aufgabe 1.1: Vervollständigen Sie das Programm mit Hilfe von Pipes. Vaterprozess und Kindprozess sollen also über Pipes miteinander kommunizieren. Sobald der Vaterprozess die richtige Zahl geraten hat, beendet er den Kindprozess mit einem `SIGKILL`-Signal. Ach-ten Sie bei der Implementierung darauf, möglichst wenige globale Variablen zu nutzen (ide-alerweise nur die Pipes).

2. MESSAGE QUEUES

Mehrere unabhängige Prozesse (mehrere Clients und ein Server) kommunizieren über eine Message-Queue.

Der Server-Prozess empfängt von den Client-Prozessen Nachrichten mit dem Message-Typ `1`. Die Client-Prozesse setzen zum Empfangen von Nachrichten vom Server-Prozess ihre eigene PID als Message-Typ ein.

Sobald der Server-Prozess die Nachricht mit der PID des startenden Client-Prozesses empfangen hat, antwortet er mit der Nachricht „Willkommen!“. An alle anderen ihm bekannten Client-Prozesse schickt er eine Nachricht „Prozess `PID` ist jetzt auch dabei“.



Der geschilderte Sachverhalt wurde Ihnen in Form eines Server- und eines Client-Programms zur Verfügung gestellt.

Aufgabe 2.1: Vervollständigen Sie die Programme mit Hilfe einer Message-Queue. Sowohl Server als auch Clients laufen in einer Endlosschleife, damit sie Nachrichten empfangen können. Die Prozesse sollen mit der Tastenkombination `CTRL+C` beendet werden können.

Aufgabe 2.2: Lassen Sie sich mit dem Befehl `ipcs -q` die existierenden Message-Queues anzeigen. In der Ausgabe sehen Sie auch, wie viele Nachrichten sich noch in der Message-Queue befinden.

Statusabfrage aller vorhandenen Message-Queues:

```
ipcs -q
```

Beispielhafte Ausgabe*:*

Schlüssel	*msqid*	*Besitzer*	*Rechte*	*BenutztBytes*	*Nachrichten*
0x00000abc	0	nn	0770	0	0

Löschen der Message-Queue:

`ipcrm -q msqid` (`0` im Beispiel)

`ipcrm -Q Schlüssel` (`0x00000abc` im Beispiel)

3. Shared Memory

Aufgabe 3.1.: Schreiben Sie drei unabhängige Programme:

- Das erste Programm erzeugt ein 20 Byte großes Shared-Memory-Segment und füllt es mit ASCII-Zeichen. Nach dem Füllen mit ASCII-Zeichen wartet das Programm auf eine beliebige Benutzereingabe (`getchar();`). Erst nachdem diese Eingabe erfolgt ist, löst das Programm das erzeugte Shared-Memory-Segment aus seinem Adressraum (ohne es zu löschen) und beendet sich.

- Das zweite Programm liest das vom ersten Programm erzeugte Shared-Memory-Segment und gibt die ASCII-Zeichen im Terminal aus.

- Das dritte Programm löscht das Shared-Memory-Segment.

Die Shared-Memory-Segmente können Sie wieder mit dem Befehl `ipcs` beobachten. Statt `ipcs -q` wie bei den Message Queues wird `ipcs -m` verwendet. Analog zu den Message Queues können Shared-Memory-Segmente mit `ipcrm -m` gelöscht werden.

Aufgabe 3.2: Beobachten sie die drei Programme mit Hilfe des Befehls `ipcs -m`. Starten Sie das erste Programm ohne eine Eingabe vorzunehmen. Starten Sie danach das dritte Programm. Welche Beobachtungen können Sie mit `ipcs -m` machen? Was passiert, wenn Sie nun auch noch das zweite Programm starten? Erklären Sie Ihre Beobachtungen.

4. SCHEDULING

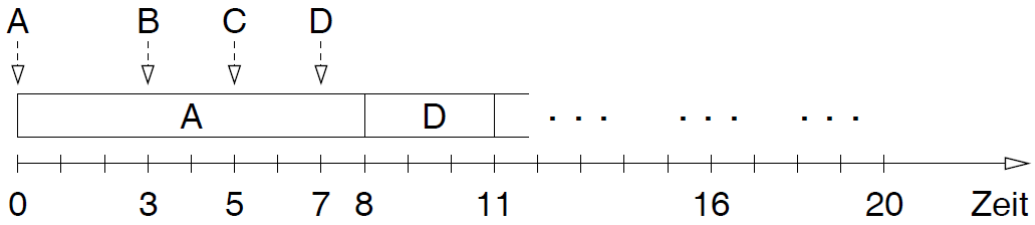
Es sei folgende Situation gegeben: Vier Threads treffen zu den angegebenen Zeiten in der Bereitliste eines Schedulers ein. Von jedem Thread ist die Bedienzeit (= benötigte Rechenzeit) und seine Priorität (0 = höchste Priorität) bekannt:

Thread	Ankunftszeit	Bedienzeit	Priorität
A	0	8	1
B	3	5	0
C	5	4	1
D	7	3	0

Die Threads geben die CPU nie freiwillig ab und werden auch nie durch E/A oder sonstige Gründe blockiert. Der Scheduler entscheidet zur Laufzeit nur aufgrund der zum aktuellen Zeitpunkt bereits eingetroffenen und lauffähigen Threads.

a) Suche nach passenden Scheduling-Verfahren

Mit einem bestimmten Scheduling-Verfahren ergibt sich für die oben beschriebene Situation der folgende (unvollständige) Zeitablauf als Gantt-Diagramm (Die Pfeile deuten an, wann die Threads im System eintreffen.):



Welche Scheduling-Verfahren könnten hier zum Einsatz gekommen sein? Kreuzen Sie die richtigen Antworten an:

- ☐ FCFS (First Come First Served)
- ☐ RR (Round-Robin) mit einer Zeitscheibenlänge (Quantum) von 4
- ☐ nicht-präemptives SJF (Shortest Job First)
- ☐ präemptives SRTF (Shortest Remaining Time First)
- ☐ präemptives prioritätsbasiertes Scheduling (HPF)

b) Präemptives prioritätsbasiertes *Feedback*-Scheduling

Geben Sie den zeitlichen Ablauf für die Ausführung der oben beschriebenen Threads als Gantt-Diagramm an, wenn präemptives prioritätsbasiertes Multilevel-Scheduling mit zwei Prioritätsklassen verwendet wird. Bei diesem Scheduling-Verfahren wird eine Prioritätsscheduling zwischen Prioritätsklassen durchgeführt. Innerhalb einer Klasse wird jeweils Round Robin (RR) verwendet, wobei die Zeitscheibenlänge (= Quantum) unterschiedlich ist:

- In der Prioritätsklasse 0 (→ hohe Priorität) ist die Zeitscheibenlänge 3;
- in der Prioritätsklasse 1 (→ niedrige Priorität) ist die Zeitscheibenlänge 4.

Ein Thread nimmt nur dann am RR-Schedulingverfahren seiner Prioritätsklasse Teil, wenn in keiner höherprioren Klasse ein Thread ablauffähig ist.

Geben Sie zu den markierten Zeitpunkten den Inhalt der beiden RR-Thread-Warteschlangen an. Der Thread, der als nächstes die CPU bekommt, soll dabei rechts stehen.

Geben Sie für alle Threads auch die Wartezeiten und Durchlaufzeiten an.