Übungsblatt 4

Abgabe bis spätestens 23.11.2023 in Stud.IP

Allgemeiner Hinweis

Für dieses Übungsblatt benötigt ihr das in StudIP zur Verfügung gestellte Archiv `ueb4-vorgabe.zip` mit Vorgabedateien.

Aufgabe 1 (5 Punkte)

Gegeben seien sechs immer lauffähige Prozesse mit den folgenden Eigenschaften (die Basispriorität ist ausschließlich für Aufgabenteil b) relevant):

Prozess	Laufzeit	Startzeitpunkt	Basispriorität
P_1	800 ms	0 ms	10
P_2	580 ms	200 ms	20
P_3	500 ms	320 ms	0
P_4	720 ms	400 ms	40
P_5	800 ms	400 ms	10
P_6	380 ms	650 ms	0

Ermittelt unter Verwendung des in `aufgabe1/CPU_Scheduling_Simulator_*.html` bereitgestellten Scheduling-Simulators den jeweiligen Prozessablauf für die folgenden Szenarien. Als Ergebnis soll Eurer Abgabe jeweils die mit der Download-Funktion erstellte Grafik unter dem in der jeweiligen Teilaufgabe angegebenen Namen beigelegt sein. Bindet die Grafiken außerdem in Euer Dokument ein.

a) *Round-Robin* mit einer Zeitscheiben-Länge von 100 ms. Begründet kurz für die ersten neun Scheduling-Vorgänge, wodurch der Vorgang ausgelöst wird und warum dem ausgewählten Prozess die CPU-Zeit zugeteilt wird. Speichert das Ergebnis in Eurer Abgabe als `aufgabe1a.png`.

b) *Kontenmodell* analog zur Vorlesung mit einer Zeitscheiben-Länge von 100 ms und den in der oben angeführten Tabelle angegebenen Basisprioriäten. Beschreibt kurz für die ersten acht Scheduling-Vorgänge, wieso dem ausgewählten Prozess die CPU zugeteilt wird und listet jeweils die Konten der Prozesse auf. Speichert das Ergebnis in Eurer Abgabe als `aufgabe1b.png`.

c) *Shortest Remaining Job First*. Begründet kurz für die ersten acht Scheduling-Vorgänge, wodurch der Vorgang ausgelöst wird und warum dem ausgewählten Prozess die CPU-Zeit zugeteilt wird. Speichert das Ergebnis in Eurer Abgabe als `aufgabe1d.png`.

d) Ermittelt für jeden Prozess und jeden der drei Algorithmen die Dauer des jeweiligen Prozesses ab dem in der obigen Tabelle genannten **Startzeitpunkt des Prozesses**. Gebt diese als Tabelle in Eurer Dokumentation mit ab. Bewertet die Ergebnisse **ausführlich** vor dem Hintergrund **interaktiver Multi-User-Systeme**.

Aufgabe 2 (5 Punkte)

Schreibt eine einfache Shell, die in der Lage ist, Kommandos mit mehreren Argumenten einzulesen und entweder im Vordergrund oder im Hintergrund zu starten. Die Einleseroutine selbst (einschließlich Parsen der Argumente) und ein Rumpf für das Hauptprogramm sind in dem Verzeichnis `aufgabe2` vorgegeben.

Unser `ti2sh` verhält sich schon fast wie eine Shell, führt aber noch keine Kommandos aus. Erweitert die Vorgabe so, dass tatsächlich Prozesse erzeugt und in ihnen die eingelesenen Kommandos ausgeführt werden, und erläutert kurz Eure Vorgehensweise. (Ihr sollt nur die Datei `ti2sh.cc` erweitern.) Hinweise dazu:

- Die typische Eingabesyntax eines Unix-Shellkommandos mit z. B. zwei Argumenten, das im Hintergrund gestartet werden soll, hat bekanntlich die Form:
  `kommando param1 param2 &`
- Die vorgegebene Einleseroutine parst die Kommandoeingabe und liefert eine Struktur `Command` mit den Komponenten `argv` und `background` zurück.
  1. argv ist ein Vektor aus C++-Strings. Der erste Eintrag in diesem Array enthält den Kommandonamen, der zweite das erste Argument des Kommandos usw. (d. h. die „Worte“ der Eingabe werden von der Einleseroutine einzeln in den Vektor geschrieben). `cmd->argv[0]` indiziert den ersten Eintrag.
  2. Wenn das angegebene Kommando nicht im Hintergrund ausgeführt werden soll, ist `background` false, andernfalls ist `background` true.
- Die zu implementierende Shell soll keine Ein-/Ausgabeumlenkung (oder sonstige weitergehende Shell-Funktionalitäten) vorsehen. Auch müsst Ihr Euch explizit nicht um im Hintergrund laufende Prozesse kümmern, die von `STDIN` lesen wollen.
- Der Systemaufruf `fork()` erzeugt aus dem aktuellen Prozess heraus einen Kindprozess. Beide Prozesse durchlaufen zunächst dasselbe weitere Programm. Ein `fork()` liefert dem Vaterprozess die Prozess-ID des Kindes zurück und dem Kindprozess den Wert `0`.
- Vergegenwärtigt Euch, was es bedeutet, einen Prozess in der Shell im Vordergrund bzw. Hintergrund zu starten, wie sich also Eure Shell zu verhalten hat und bedenkt, dass dies zunächst keine Auswirkungen auf den Zustand des Prozesses hat.
- Der Systemaufruf `wait()` zum Warten auf die Termination der von Eurer Shell erzeugten Kindprozesse liefert die Prozess-ID des terminierten Prozesses zurück und benötigt einen Parameter, den wir hier nicht nutzen wollen und deshalb auf `0` setzen (ein Nullpointer):
  `terminated_child = wait(0);`
  Alternativ könnt ihr `waitpid()` nutzen. Je nach Implementierungsansatz werdet Ihr einen Signal-Handler für das Signal `SIGCHLD` anmelden müssen, um tatsächlich alle Kinder einsammeln zu können. Benutzt in diesem Fall `sigaction` zum Einrichten des Signalhandlers und setzt das Flag `SA_RESTART`.
- Systemaufrufe liefern im Fehlerfall oftmals `-1` zurück. Die Funktionen `perror()` und `strerror()` ermöglichen es dann, sinnvolle Fehlermeldungen auszugeben.
- Verwendet `ti2::searchCommand()`, um zu einem Kommando einen Kandidaten für das auszuführende Programm zu finden. Diese Funktion erwartet zwei Parameter: Den Namen des auszuführenden Programms und einen Ergebnisparameter, der im Erfolgsfall mit dem Namen des an `execv()` zu übergebenden Programms befüllt wird.

- Es ist nur die mitgelieferte Version von `execv()` zulässig, keine andere Variante von `exec()`.

Testet Eure Implementierung **angemessen und ausführlich**, d. h. nehmt Eingaben vor, die die geforderten Eigenschaften demonstrieren und **dokumentiert** dies. Berücksichtigt typische Fehlerfälle.

Abgabe

Bis 23:59 Uhr am 23.11.2023 digital in StudIP. **Es gelten die vereinbarten Scheinbedingungen (siehe StudIP).** Bitte beachtet unsere ergänzenden Hinweise ebenda.

Ladet die abgaberelevanten Dateien in DoIT hoch.

Eure Ansätze und der gewählte Lösungsweg müssen nachvollziehbar sein. Achtet insofern auf eine saubere Dokumentation im Quelltext und im abgegebenen PDF-Dokument. Benennt alle von Euch verwendeten Quellen, auch Zusammenarbeit mit anderen Gruppen und verwendete Unterlagen aus früheren Jahrgängen.

Programmieraufgaben sind im Zweifel in C++20 zu entwickeln. Die Korrektheit der Lösung bzw. deren Grenzen sind grundsätzlich in der Abgabe nachzuweisen. Dies geschieht neben der Dokumentation des Programmcodes in den Quelldateien und zusätzlich in dem abgegebenen PDF-Dokument mit der Lösungsbeschreibung durch geeignete Tests, deren Auswahl und Eignung begründet werden müssen. Als **Referenzplattform** gelten die Linux-Rechner im Rechnerpool in MZH E0.