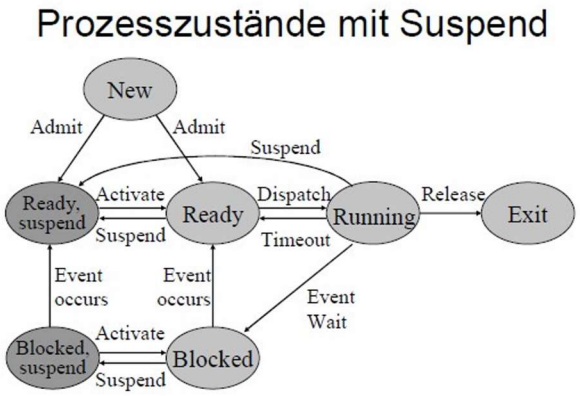
Prozesse, Threads, Tasks

- Prozesse definieren ein ausführbares Programm.
- Jeder Prozess besitzt eine eindeutige **Prozess ID**, Daten (Stack, Heap, …) und einen aktuellen Prozesskontext (aktueller Zustand des Prozesses und Daten zur Prozessverwaltung(Priorität des Prozesses, Wartebedingung, …))
- Jeder Prozess kann während seiner Lebensdauer seinen Zustand mehrere Male ändern.
- Die Verwaltung vom Prozess hängt vom Betriebssytem ab.
- Verwaltung zum Beispiel mit Hilfe von Queues. Bei jeder Zustandsänderung wird der Prozess in eine andere Queue verschoben.

Prozesse, Threads, Tasks



Prozesszustände

New
– BS hat den Prozess kreiert:
Prozessnummer (process identifier)
Tabellen und Tabelleneinträge zur Prozessverwaltung
– der Prozess ist jedoch noch nicht bereit zur Ausführung
Vermeidung der Ressourcenüberlastung durch Zulassung zuvieler Prozesse
Running
– Prozess ist im Besitz der CPU und wird ausgeführt
• *Ready*
– Prozess ist bereit zur Ausführung (wartet nur auf Zuteilung der CPU)
• *Blocked*
– Prozess wartet auf ein Ereignis (z.B. Beendigung von I/O), ist nicht laufbereit

Prozesszustände

Exit
– Zustand wird durch Terminierung erreicht
– Prozess wird nicht mehr weiter ausgeführt
– Prozessinformationen (Tabellen) werden von Hilfsprogrammen verwendet (z.B. Accounting)
– Die Prozessinformationen, -tabellen werden gelöscht, wenn sie nicht mehr benötigt werden

Suspend

– Eine Art Wartezustand für Prozesse.
– Zustand entsteht beim Swapping (Prozesse werden aus dem Hauptspeicher auf die Festplatte augelagert)

Threads

- Threads sind „lightweighted" Prozesse
- Mehrere Threads pro Prozess möglich
- Zugriff auf den gleichen Adressraum eines Prozesses
- Ausführungszustände Running, Ready, Blocked
- Bei Terminierung eines Prozesses terminieren auch alle dem Prozess angehörige Threads
- Verschiedene Threads eines Prozesses teilen sich einen Speicherbereich
- Ein Thread ist eigentlich der Teil, den eine CPU ausführt

Scheduler & Dispatcher

- Der Scheduler ist eine Software bzw. Teil des Betriebssystems.
- Der Scheduler entscheidet welcher Prozess als nächster die CPU zugeordnet bekommt, um Operationen durchzuführen.
- Der Dispatcher ist ebenfalls eine Software und auch Teil des Betiebssystems
- Der Dispatcher gibt einem Prozess die Kontrolle über die CPU und verschiebt den Prozess aus einem „Ready" Zustand in einen Ausführungszustand.

Klasse Thread / InterfaceRunnable

- In Java gibt es 2 Möglichkeiten einen Thread zu erzeugen.
- Entweder man erbt aus der Klasse Thread und implementiert die Funktion run()…
- .. oder man implementiert das Interface Runnable und implementiert dort ebenfalls die Funktion run()

Gemeinsame Aufgabe– Multitasking Bot

Entwickle ein Programm für einen Bot, dass es ihm ermöglicht gleichzeitig die Zahlen von 1 - 1000 als auch die Buchstaben von A -Z auszugeben.

Inter-thread Synchronisation

- Synchronisation zwischen Threads ist notwendig, um es möglich zu machen, dass ein Thread in einem kritischen Codeteil pausiert und ein anderer Thread diesen kritischen Codeteil betritt
- Um die Synchronisation zwischen mehreren Threads zu ermöglichen sind 3 Funktionen gegeben: wait(), notify(), notifyAll()
- Um diese Funktionen zu verwenden, muss die jeweilige Funktion, in der diese Funktionen aufgerufen werden mit dem Schlüsselwort synchronized deklariert werden.
- wait() versetzt einen Thread in einen pausierenden Zustand
- notify() erweckt einen Thread, der im selben Objekt in den pausierenden Zustand versetzt wurde
- notifyAll() erweckt alle Threads
- join() ist eine Funktion mit der man explizit auf einen Thread warten kann.

Zugriff auf Ressourcen synchronisieren

- Es kann unter Umständen vorkommen, dass mehrere Threads auf eine Ressource zugreifen wollen – es könnten mehrere Threads zum Beispiel auf ein File schreiben wollen
- Damit sich Threads nicht gegenseitig beim Schreiben stören, muss der Zugriff auf das File synchronisiert werden
- Mit dem Schlüsselwort synchronized (ACHTUNG, das ist ein anderes synchronized als bei Funktionen) kann man ein gemeinsam genutztes Objekt in einen synchronisierten Codeblock packen, sodass nur ein Thread auf dieses Objekt zugreifen kann

Deadlock

- Wenn ein Thread auf eine Ressource wartet, dass von einem anderen Thread gehalten wird und vice verca, kann es sein, dass ein sogenannter Deadlock entsteht
- Deadlock bedeutet, dass beide Threads die Ressourcen nicht freigeben können, weil diese auf die Ressource des anderen Thread warten.
- Deadlock Beispiel:

Lösung (Reihenfolge des Locks ändern):

Gemeinsames Beispiel- Reservierung

Du entwickelst ein einfaches Ticket -Reservierungssystem für ein Theater. Das Theater hat eine begrenzte Anzahl von Sitzplätzen, und mehrere Benutzer versuchen gleichzeitig Tickets zu reservieren. Stelle sicher, dass keine Überbuchungen oder Konflikte auftreten.

Implementiere in einer Klasse eine Funktion makeReservation, in der Ein Benutzer eine Reservierung vornehmen kann. Dabei kann er wählen, wie viele Sitzplätze er benötigt.

Implementiere eine Funktion cancelReservation, die eine Reservierung wieder storniert und die Sitzplätze wieder freigibt.

Pipes

- Pipes sind eine Möglichkeit für Threads miteinander zu kommunizieren.
- Dabei werden die Klassen PipedOutputStream und PipedInputStream verwendet


PipeExample.java