Aufgabe V7: Semaphore für Erzeuger/Verbraucher

Eine Pufferverwaltung zur Datenflusskontrolle unterscheidet zwischen leeren und gefüllten Puffern. Alle Puffer haben die gleiche Größe. Zwei Arten von Prozessen nutzen diese Verwaltung: *Erzeuger* fordern leere Puffer an und geben gefüllte zurück, *Verbraucher* fordern gefüllte Puffer an und geben leere zurück.
Die Puffer sind ab 0 durchnummeriert, ihre Gesamtzahl wird einmalig bei der Initialisierung festgelegt. Die Verwaltung verwendet zwei Bitvektoren entsprechender Länge. Der Bitvektor RB gibt für jeden Puffer an, ob dieser in Ruhe (0) oder in Bearbeitung (1) ist. Der Bitvektor LG gibt für jeden Puffer in Ruhe an, ob er leer (0) oder gefüllt (1) ist. Die Operationen laufen wie folgt ab:

Leer Anfordern: *kein Argument, Rückgabewert ist eine Puffernummer*

1. einen Puffer suchen, für den RB und LG auf 0 stehen
2. entsprechendes Bit in RB auf 1 setzen
3. entsprechende Puffernummer zurückgeben

Voll Zurückgeben: *Argument ist eine Puffernummer, kein Rückgabewert*

1. entsprechende Bits in RB auf 0 und in LG auf 1 setzen

Voll Anfordern: *kein Argument, Rückgabewert ist eine Puffernummer*

1. einen Puffer suchen, für den RB auf 0 und LG auf 1 steht
2. entsprechendes Bit in RB auf 1 setzen
3. entsprechende Puffernummer zurückgeben

Leer Zurückgeben: *Argument ist eine Puffernummer, kein Rückgabewert*

1. entsprechende Bits in RB und LG auf 0 setzen

In diesen Abläufen fehlt noch die Synchronisation der Prozesse. Gehen Sie davon aus, dass diese nur gültige Aufrufe durchführen, also nur zuvor angeforderte Puffer zurückgeben. Das Anfordern wartet jeweils, bis ein entsprechender Puffer verfügbar ist.

a) Die Verwaltung nutzt drei Semaphore `M`, `L` und `G`. Mit welchen Werten werden diese initialisiert? Wo finden die Aufrufe von `P()` bzw. `V()` statt?

Die Namen der Semaphore sind ein Hinweis auf ihre vorgesehene Verwendung: Mutex, Leer, Gefüllt. Man könnte die Rollen natürlich auch anders zuordnen.

Wie in der vorhergehenden Aufgabe „Semaphore im Einsatz“ ist `M` eine Sperre und wird mit 1 initialisiert. Sie sichert den exklusiven Zugriff auf beide Bitvektoren. Da das Anfordern immer Zugriff auf beide braucht, ergäbe es keinen Sinn, für jeden Bitvektor eine eigene Sperre vorzusehen.

`L` dient als Zähler für leere und `G` für gefüllte Puffer. Bei der Initialisierung sind alle Puffer leer. Also wird `L` mit der Anzahl der Puffer und `G` mit 0 initialisiert.

Leer Anfordern: zu Beginn erst `L.P()` dann `M.P()`, nach Schritt 2 `M.V()`

Voll Zurückgeben: zu Beginn `M.P()`; am Ende `M.V()` und `G.V()`

Voll Anfordern: zu Beginn erst `G.P()` dann `M.P()`, nach Schritt 2 `M.V()`

Leer Zurückgeben: zu Beginn `M.P()`, am Ende `M.V()` und `L.V()`

b) Ist sichergestellt, dass jeder gefüllte Puffer auch wieder geleert wird, sofern Erzeuger und Verbraucher ihren Aufgaben nachkommen? Falls nicht, was muss man ändern?

Nein, es ist nicht sichergestellt. Die Reihenfolge, in der gefüllte Puffer beim Anfordern gewählt werden, ist nicht festgelegt. Falls die Erzeuger schnell genug sind, ständig mehrere gefüllte Puffer bereitzustellen, könnten Puffer „verhungern“. Das passiert, wenn die Verwaltung den Verbrauchern immer wieder frisch gefüllte Puffer ausgibt, aber ältere gefüllte Puffer liegen lässt. Zum Beispiel bei einer Suchstrategie, die jeweils den gefüllten Puffer mit der niedrigsten Nummer findet.

Um dieses Problem zu beheben, braucht man eine definierte Suchstrategie und zusätzliche Informationen in der Pufferverwaltung. Zum Beispiel könnte man die gefüllten Puffer zusätzlich in einer Liste vermerken und in der gleichen Reihenfolge ausgeben, in der sie gefüllt wurden (engl.: *FIFO — first in, first out*). Oder man merkt sich die Nummer des zuletzt ausgegebenen gefüllten Puffers und fängt bei der nächsten Suche mit dem darauf folgenden an.

Bei den leeren Puffer könnte es ebenfalls zu ungleichmäßigen Belegungen kommen. Aber dort spielt es keine Rolle, weil die Puffer ja leer sind.

c) In Fehlersituationen kann es notwendig sein, dass Erzeuger einen leeren Puffer zurückgeben, oder Verbraucher einen gefüllten. Funktioniert das mit den beschriebenen Abläufen und der Synchronisation? Falls nicht, was muss man ändern?

Ja, es funktioniert. Die Operationen prüfen nicht ab, ob sie „wie vorgesehen“ paarweise verwendet werden. Mit den Abläufen und der Synchronisation wie oben beschrieben bleiben die Verwaltungsdaten auch in den ungewohnten Kombinationen konsistent.

Wenn die Operationen auf ihre Verwendung prüfen würden, zum Beispiel um Programmierfehler zu erkennen, müsste man etwas ändern. Entweder die Prüfungen entfernen, oder zusätzliche Wege für das Zurückgeben im Fehlerfall anbieten. Letzteres ginge mit zwei weiteren Operationen, oder mit einem zusätzlichen Parameter bei den existierenden.

Ob man solche Prüfungen durchführt, und wie man die Operationen gestaltet, gehört zu den freien Entscheidungen beim Entwurf von Programmierschnittstellen. Richtig oder falsch gibt es dabei nicht, nur Abwägungen zwischen vielen Faktoren: Nutzerfreundlichkeit, Flexibilität, Komplexität, Geschwindigkeit,...