

Musterlösungen zu den Tutoraufgaben auf
Blatt 6
der Übungen zur Vorlesung
“Grundlagen Betriebssysteme und Systemsoftware

G.Groh, 20.11.2008

Aufgabe 4

```
final int N=100 /* Anzahl Plätze im Puffer */
int count = 0; /* Anzahl der Elemente im Puffer */
void producer(void) {
    int item;
    while (TRUE) { /* Endlosschleife */
        item = produce_item(); /* erzeuge nächstes Element */
        if (count == N)
            sleep(); /* wenn Puffer voll, schlafe */
        insert_item(item); /* schreibe Element in Puffer */
        count = count + 1; /* erhöhe Anzahl der Elemente */
        if (count == 1)
            wakeup(consumer); /* war der Puffer leer? */
    }
}
void consumer(void) {
    int item;
    while (TRUE) { /* Endlosschleife */
        if (count == 0)
            sleep; /* wenn Puffer leer, schlafe */
        item = remove_item(); /* hole Element aus Puffer */
        count = count - 1; /* erniedrige Anzahl der Elemente */
        if (count == N-1)
            wakeup(producer); /* war der Puffer voll? */
        consume_item(item);
    }
}
```

Aufgabe 4

```
final int N=100 /* Anzahl Plätze im Puffer */
int count = 0; /* Anzahl der Elemente im Puffer */
void producer(void) {
    int item;
    while (TRUE) { /* Endlosschleife */
        item = produce_item(); /* erzeuge nächstes Element */
        if (count == N)
            sleep(); /* wenn Puffer voll, schlafe */
        insert_item(item); /* schreibe Element in Puffer */
        count = count + 1; /* erhöhe Anzahl der Elemente */
        if (count == 1)
            wakeup(consumer); /* war der Puffer leer? */
    }
}

void consumer(void) {
    int item;
    while (TRUE) { /* Endlosschleife */
        if (count == 0)
            sleep; /* wenn Puffer leer, schlafe */
        item = remove_item(); /* hole Element aus Puffer */
        count = count - 1; /* erniedrige Anzahl der Elemente */
        if (count == N-1)
            wakeup(producer); /* war der Puffer voll? */
        consume_item(item);
    }
}
```

Unterbrechung hier →
Thread Interference
möglich.

Wie wahrscheinlich ist
so etwas?

Kann das in der
Implementierung der
JVM auftreten?

Zugriff auf count
„unconstrained“ →
Memory Inconsistency
Error möglich

Aufgabe 4

```
monitor ErzeugerVerbraucher
condition full, empty;
integer count;
procedure insert(item: integer);
begin
    if count = N then wait(full);
    insert_item(item);
    count := count + 1;
    if count = 1 then signal(empty)
end;
function remove: integer;
begin
    if count = 0 then wait(empty);
    remove = remove_item;
    count := count - 1;
    if count = N - 1 then signal(full)
end;
count := 0;
end monitor;

procedure Erzeuger;
begin
    while true do
    begin
        item = produce_item;
        ErzeugerVerbraucher.insert(item)
    end
end;
procedure Verbraucher;
begin
    while true do
    begin
        item = ErzeugerVerbraucher.remove;
        consume_item(item)
    end
end;
end;
```

Aufgabe 4

```
monitor ErzeugerVerbraucher
condition full, empty;
integer count;
procedure insert(item: integer);
begin
    if count = N then wait(full);
    insert_item(item);
    count := count + 1;
    if count = 1 then signal(empty)
end;
function remove: integer;
begin
    if count = 0 then wait(empty);
    remove = remove_item;
    count := count - 1;
    if count = N - 1 then signal(full)
end;
count := 0;
end monitor;

procedure Erzeuger;
begin
    while true do
    begin
        item = produce_item;
        ErzeugerVerbraucher.insert(item)
    end
end;

procedure Verbraucher;
begin
    while true do
    begin
        item = ErzeugerVerbraucher.remove;
        consume_item(item)
    end
end;
end;
```

Ggf Blockieren

Zwar wäre hier theoretisch zunächst eine Unterbrechung möglich, aber da der andere Thread den Monitor nicht betreten darf kann es weder zum Fehlertyp „Thread-Interference“ kommen. Es kann aus dem gleichen Grund (anderer Thread kann nicht rein) auch keine „Memory Inconsistency“ geben.

Aufgabe 4

Lösung mit Java

(Quelle: Tanenbaum (deutsche Ausgabe))

```
public class ProducerConsumer {
    static final int N = 100; // Puffergröße
    static producer p = new producer(); // erzeuge neuen Erzeuger-Thread
    static consumer c = new consumer(); // erzeuge neuen Verbraucher-Thread
    static out_monitor mon = new out_monitor(); // erzeuge neuen Monitor
    public static void main(String args[]) {
        p.start(); // starte Erzeuger-Thread
        c.start(); // starte Verbraucher-Thread
    }
    static class producer extends Thread {
        public void run() { // in der run-Methode liegt der Threadcode
            int item;
            while (true) { // Schleife des Verbrauchers
                item = produce_item();
                mon.insert(item);
            }
            private int produce_item() {...} // erzeuge etwas
        }
    }
    static class consumer extends Thread {
        public void run() { // in der run-Methode liegt der Threadcode
            int item;
            while (true) { // Schleife des Verbrauchers
                item = mon.remove();
                consume_item(item);
            }
            private void consume_item(int item) {...} // verbrauche etwas
        }
    }
    static class out_monitor { // das ist der Monitor
        private int buffer[] = new int[N];
        private int count = 0, lo = 0, hi = 0; // Zähler und Indizes
        public synchronized void insert(int val) {
            if (count == N) go_to_sleep(); // wenn Puffer voll, schlafe
            buffer [hi] = val; // lege Element in den Puffer
            hi = (hi+1) % N; // Platz für nächstes Element
            count = count + 1; // ein Element mehr im Puffer
            if (count == 1) notify(); // wenn Verbraucher schläft, aufwecken
        }
        public synchronized int remove() {
            int val;
            if (count == 0) go_to_sleep(); // wenn Puffer leer ist, schlafe
            val = buffer [lo]; // hole Element aus dem Puffer
            lo = (lo + 1) % N; // Platz für nächstes Element
            count = count - 1; // ein Element weniger im Puffer
            if (count == N - 1) notify(); // wenn Produzent schläft, aufwecken
            return val;
        }
        private void go_to_sleep() {
            try { wait (); }
            catch (InterruptedException exc) { };
        }
    }
}
```