Institut für Technische Informatik

Entwurf von Echtzeitsystemen

Wintersemester 2014

6. Übungsblatt

Übungsdatum: 19.12.2014

Gruppe: 1

Ausarbeitung von: Martin Winter

Wildon, am 17. Dezember 2014

Tasksynchronisations (Locking)

Warum und wann ist Tasksynchronisation nötig und was bedeutet Atomizität und Kritikalität in diesem Zusammenhang? Erklären Sie das Problem anhand des folgenden Beispiels.
Führt dieses Beispiel in jedem Fall zur korrekten Realisierung einer kritischen bzw. atomaren Sektion und wird dabei aktives Warten oder passives Warten verwendet? Wann ist es sinnvoll, Spinlocks einzusetzen, wann nicht,m und wie müsste der code geändert werden, um die jeweils andere Variante zu realisieren?
Kann es unter Verwendung des gezeigten Konzepts durch mehrere Tasks zu Lifelocks oder Deadlocks kommen? Skizzieren Sie gegebenenfalls jeweils eine entsprechendes Szenario.

Tasksynchronisation (Producer-Consumer-Problem)

Erinnern Sie sich zurück an das in der Vorlesung besprochenen Ereignissynchronisationsproblem: Ein Monitortask überwacht darin den Systemzustand und sendet bei Überschreitung von Grenzwerten Nachrichten an einen Kommunikationstask. Dieser werden in einem Puffer zwischengespeichert.
Welche Problematik kann dazu führen, dass eine gepufferte Nachricht vom Kommunikationstask nicht weitergeleitet wird? Beschreiben Sie diese anhand des Beispiels aus der Vorlesung genauer!
Das zuvor betrachtete Problem enthält das Erzeuger-Verbraucher-Problem, beschreiben Sie dieses genauer und präsentieren sie Lösungsvarianten dazu!

Ich kann leider bei bestem Willen dieses Problem nicht finden, das in der Vorlesung besprochen wurde, anscheinend! In den Folien finde ich nix dazu und in die blöde VO geh ich ja net, ist mir zu schade! Hier ist aber die Lösung des ProCon-Problems :)

```c
#define BUFF_SIZE   5   /* total number of slots */
#define NP          1   /* total number of producers */
#define NC          1   /* total number of consumers */
#define NITERS      10  /* number of items produced/consumed */
sem_t spaces;           /* number of available spaces in buffer */
sem_t items;            /* number of items */
pthread_mutex_t mutex;  /* mutual exclusion to shared data */

int main()
{
    pthread_t idP, idC;
    int index;

    sem_init(&items, 0, 0);
    sem_init(&spaces, 0, BUFF_SIZE);
    pthread_mutex_init(&mutex, NULL);

    for (index = 0; index < NP; index++)
    {
        /* Create a new producer */
        pthread_create(&idP, NULL, Producer, (void*)index);
    }

    for(index=0; index<NC; index++)
    {
      /*create a new Consumer*/
      pthread_create(&idC, NULL, Consumer, (void*)index);
    }
    // Join on all threads in the end!
    return 0;
}
```

Listing 1: code/main.c

```c
void *Producer(void *arg)
{
    int i, item, index;

    index = (int)arg;
    for (i=0; i < NITERS; i++)
    {
        item = i;                       // Produce an item

        sem_wait(&spaces);              // Decrement the number of free spaces
        pthread_mutex_lock(&mutex);     // Get access to shared data

        shared.buf[shared.in] = item;
        shared.in = (shared.in+1)%BUFF_SIZE;
        printf("[P%d] Producing %d ...\n", index, item);

        pthread_mutex_unlock(&mutex);   // Release mutex to shared data
        sem_post(&items);               // increment Number of items
        sleep(1);
    }
    return NULL;
}
```

Listing 2: code/producer.c

```c
void *Consumer(void *arg)
{
    int i, item, index;

    index = (int)arg;
    for (i = NITERS; i > 0; i--) {
        sem_wait(&items);                   // Wait for an item
        pthread_mutex_lock(&mutex);         // Get access to shared data

        item = i;
        item = shared.buf[shared.out];
        shared.out = (shared.out+1)%BUFF_SIZE;
        printf("[C%d] Consuming %d ...\n", index, item);


        pthread_mutex_unlock(&mutex);       // Release mutex to shared data
        sem_post(&spaces);                  // increment Number of free spaces

        sleep(1);
    }
    return NULL;
}
```

Listing 3: code/consumer.c

Echtzeit-Programmiersprachen

Beschäftigen Sie sich mit den in der Vorlesung gezeigten sechs speziellen Anforderung an Echtzeit-Programmiersprachen. Beschreiben Sie die einzelnen Anforderungen und zeigen Sie, welche Bedeutung den einzelnen Anforderungen in der Realität zukommt. Überlegen Sie sich hierzu ein fiktives Beispiel und zeigen Sie, welche Auswirkung das Fehlen jeder speziellen Anforderung haben kann.

Aktivitätsorientierte Modelle

Sie nehmen mit einer Digitalkamera ein Bild auf. Dieses Bild wird in der Kamera geringfügig bearbeitet, z.B.: Reduktion roter Augen, Entzerrung, Weißabgleich, etc. Stellen Sie den Ablauf von der Aufnahme des Bildes über die Bearbeitung bis hin zur anzeige am Bildschirm als Datenflussgraph und Flussdiagramm dar. Sie können die Optimierungen, die am Bild vorgenommen werden, selbst wählen, es müssen aber mindestens 4 sein.

Multicore Scheduling

Abbildung 5 zeigt die Skizze eines Programmablaufs eines echtzeitfähigen Programms mit einer Zykluszeit von 10 ms. Um die Echtzeitfähigkeit des Programms zu garantieren, müssen alle Programmteile A bis J vor erreichen der Deadline abgeschlossen werden.

- Skizzieren Sie den Programmablauf auf einer CPU bzw. mehreren CPU's.
- Wie viele CPU's werden benötigt?
- Wie viele CPU's machen Sinn?