

Übung zu Betriebssysteme

Aufgabe 4: Threadumschaltung

Wintersemester 2021/22

Bernhard Heinloth

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme
und Betriebssysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

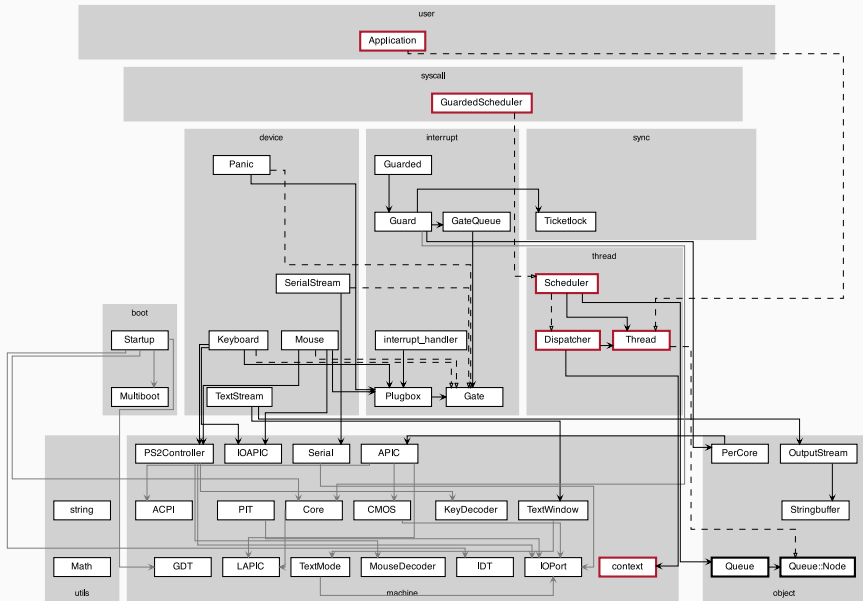
TECHNISCHE FAKULTÄT

Lernziele

- Auffrischen der Assemblerkenntnisse
- Verständnis der Abläufe beim Threadwechsel
- Unterscheiden von aktiven & passiven Objekten

Umsetzung

- Kontextwechsel
- Threadverwaltung
- Auslagerung von Anwendungen



Threads

```
class Thread {  
    StackPointer sp;  
public:  
    Thread();  
    virtual void action() = 0;  
};
```

Adresse einer virtuellen Member-Funktion nicht (einfach) ermittelbar

```
class Foo : public Thread {  
    void action() { ... }  
};  
  
class Bar : public Thread {  
    void action() { ... }  
};
```

```
Thread * x = &app;  
x->action(); // Foo::action oder Bar::action ?
```

Threads

```
class Thread {  
    StackPointer sp;  
public:  
    Thread();  
    virtual void action() = 0;  
};
```

Adresse einer virtuellen Member-Funktion nicht (einfach) ermittelbar

```
void kickoff(Thread* t){  
    t->action();  
}
```

Threads: Stack aufsetzen

```
void * prepareContext(void * tos,  
                     void (*kickoff)(void*),  
                     void * param);
```

Präpariert einen Stack für den ersten Einsprung

- statisch übergebenen Speicher `tos` als Stack aufsetzen
- nach dem Einsprung soll `kickoff` mit `param` aufgerufen werden
- Stackpointer `kernel` in Thread entsprechend setzen
- in C++ (statt Assembler)
- Pointerarithmetik ist hilfreich

```

#define STACKSIZE 4096
char stackBar[STACKSIZE];
char stackFoo[STACKSIZE];

void* tos = stackFoo + STACKSIZE;
void** rsp = (void**) tos;

rsp--;
// rsp = &(stackFoo[4088])
*rsp = (void*) 0xdeadbeef0badf00d;

rsp--;
// rsp = &(stackFoo[4080])
extern Thread foo;
// &foo = 0x102 4280
*rsp = (void*) &foo;

// ...

```

stackBar[1] →	...	0102 6021
stackBar[0] →		0102 6020
stackFoo[4095] →	de	0102 601f
stackFoo[4094] →	ad	0102 601e
stackFoo[4093] →	be	0102 601d
stackFoo[4092] →	ef	0102 601c
stackFoo[4091] →	0b	0102 601b
stackFoo[4090] →	ad	0102 601a
stackFoo[4089] →	f0	0102 6019
stackFoo[4088] →	0d	0102 6018
stackFoo[4087] →	00	0102 6017
stackFoo[4086] →	00	0102 6016
stackFoo[4085] →	00	0102 6015
stackFoo[4084] →	00	0102 6014
stackFoo[4083] →	01	0102 6013
stackFoo[4082] →	02	0102 6012
stackFoo[4081] →	42	0102 6011
stackFoo[4080] →	80	0102 6010
stackFoo[4079] →		0102 600f
	...	
stackFoo[1] →		0102 5021
stackFoo[0] →		0102 5020
	...	



Stapelüberlauf

- Die Stacks sind nur 4K groß
 - Die Stacks liegen (oft) hintereinander
 - Das gilt auch für die initialen Stacks beim Systemstart
- Mechanismus zum Erkennen von Überlaufen hilfreich („Stack Canary“)

```

#define STACKSIZE 4096
char stackBar[STACKSIZE];
char stackFoo[STACKSIZE];

void* tos = stackFoo + STACKSIZE;
void** rsp = (void**) tos;

rsp--;
// rsp = &(stackFoo[4088])
*rsp = (void*) 0xdeadbeef0badf00d;

rsp--;
// rsp = &(stackFoo[4080])
extern Thread foo;
// &foo = 0x102 4280
*rsp = (void*) &foo;

// ...

```

...		
stackBar[1] →	aa	0102 6021
stackBar[0] →	55	0102 6020
stackFoo[4095] →	de	0102 601f
stackFoo[4094] →	ad	0102 601e
stackFoo[4093] →	be	0102 601d
stackFoo[4092] →	ef	0102 601c
stackFoo[4091] →	0b	0102 601b
stackFoo[4090] →	ad	0102 601a
stackFoo[4089] →	f0	0102 6019
stackFoo[4088] →	0d	0102 6018
stackFoo[4087] →	00	0102 6017
stackFoo[4086] →	00	0102 6016
stackFoo[4085] →	00	0102 6015
stackFoo[4084] →	00	0102 6014
stackFoo[4083] →	01	0102 6013
stackFoo[4082] →	02	0102 6012
stackFoo[4081] →	42	0102 6011
stackFoo[4080] →	80	0102 6010
stackFoo[4079] →		0102 600f
	...	
stackFoo[1] →	aa	0102 5021
stackFoo[0] →	55	0102 5020
	...	

Scheduler

Scheduler verwaltet Koroutinen (Threads) zentral

```
class Scheduler {  
    Queue<Thread> readylist;  
public:  
    void ready(Thread *);  
    void schedule();  
    void resume();  
    void exit();  
    void kill(Thread *);  
};
```

```
// Thread AppFoo
void AppFoo::action(){
    while (1){
        kout << "foo" << endl;
        scheduler.resume();
    }
}
```

```
// Thread AppBar
void AppBar::action(){
    while (1){
        kout << "bar" << endl;
        scheduler.resume();
    }
}
```

```
// main.cc
AppFoo appfoo;
AppBar appbar;
int main (){
    // ...
    scheduler.ready(&appfoo);
    scheduler.ready(&appbar);
    scheduler.schedule();
}
```

main()

```
// ...  
scheduler.ready(&appfoo)  
scheduler.ready(&appbar)  
scheduler.schedule()  
StackPointer dummy;  
context_switch(dummy, appfoo.sp)
```

AppFoo

```
kickoff(&appfoo)  
appfoo->action()  
kout << "foo" << endl  
scheduler.resume()
```

context_switch(appfoo.sp, appbar.sp)

AppBar

```
kickoff(&appbar)  
appbar->action()  
kout << "bar" << endl  
scheduler.resume()
```

context_switch(appbar.sp, appfoo.sp)

```
kout << "foo" << endl  
scheduler.resume()
```

context_switch(appfoo.sp, appbar.sp)

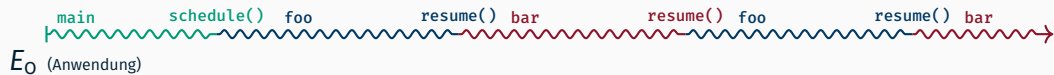
context_switch(appbar.sp, appfoo.sp)

foo

bar

foo

bar



Scheduler

Scheduler verwaltet Koroutinen (Threads) zentral

```
class Scheduler {  
    Queue<Thread> readylist;  
public:  
    void ready(Thread *);  
    void schedule();  
    void resume();  
    void exit();  
    void kill(Thread *);  
};
```

OOSTuBS immer synchrone Aufrufe

→ Konsistenz gesichert

MPStuBS Synchronisation notwendig

main()

//...

guard.enter()

scheduler.schedule()

StackPointer dummy;

context_switch(dummy, appfoo.sp)

AppFoo

kickoff(&appfoo)

guard.leave()

appfoo->action()

kout << "foo" << endl

guard.enter()

scheduler.resume()

context_switch(appfoo.sp, appbar.sp)

context_switch(appbar.sp, appfoo.sp)

guard.leave()

kout << "foo" << endl

guard.enter()

scheduler.resume()

context_switch(appfoo.sp, appbar.sp)

AppBar

kickoff(&appbar)

guard.leave()

appbar->action()

kout << "bar" << endl

guard.enter()

scheduler.resume()

guard.leave()

kout << "bar" << endl

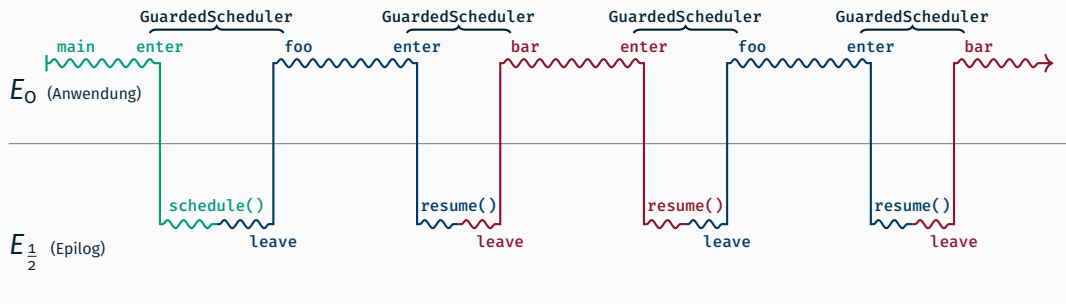
foo

bar

foo

bar

- Scheduling auf der Epilogebeene
(→ automatische Synchronisation der Ready-Liste)
- Kapselung in GuardedScheduler
(→ „Systemaufruf“ für Anwendungen)



E_1 (IRQ/Prolog)