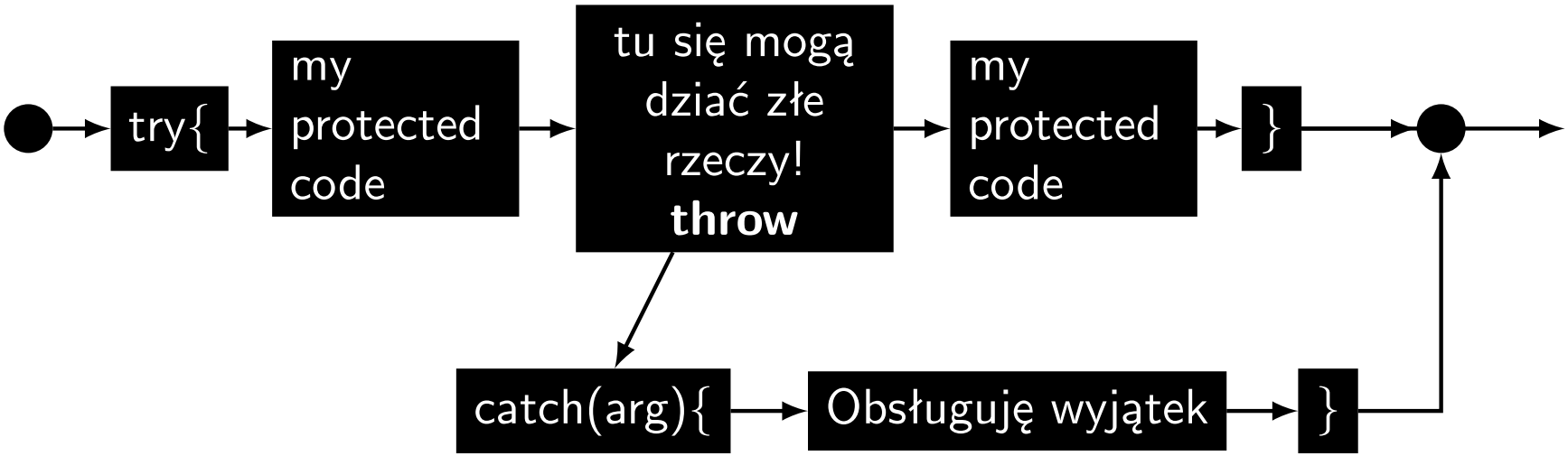
Programowanie obiektowe w języku C++

Stanisław Gepner

sgepner@meil.pw.edu.pl

Wyjątki

- Metoda przeniesienia kontroli do w przypadku wystąpienia wyjątkowych sytuacji
- Wykorzystujemy blok *try{}catch(){}*
- Wyjątek podniesiony w sekcji *try* przenosi kontrolę do odpowiedniej sekcji *catch()*



Wyjątki

- można rzucać prawie wszystkim
- a łapać, określony typ, ale nie tylko

```
#include <iostream>
using namespace std;

int main()
{
  try
  {
    throw 2;
  }
  catch (int e)
  {
    cout << "An␣exception␣occurred.␣Exception␣Nr.␣" << e << '\n';
  }
  return 0;
}
```

Wyjątki

- *catch(...){* złapie 'co kolwiek'

```
#include <iostream>
#include <string>
using namespace std;
int main(){
  try  {
   int a;
   cin >> a;
   switch(a)
   {
    case 1:
     throw 5;
    case 2:
     throw 'a';
    case 3:
     throw string("String");
    case 4:
     throw 5.0;
   }
  }
  catch (int e){ cout << "An int: " << e <<endl;}
  catch (char e){ cout << "Char: " << e << endl;}
  catch (string e){ cout << "String " << e << endl;}
  catch (...){ cout << " Default exception!!" << endl;}
  return 0;}
```

Wyjątki

- Można zagnieżdżać

```
try{
 try{
  //Code that does some serious stuff
  throw 4;
 }
 catch (int e){
  cout << "App thrown an exeption , terhrowing!" <<endl;
  ostringstream os;
  os << "Value is: " << e;
  throw os.str();
 }
}
catch (string e)
{
  cout << "An exception occurred. String " << e << endl;
}
catch (...)//domyslnie
{
 cout << " Default exception!!" << endl;
}
```

exception

- *include* < *exception* >

```
class exception {
public:
  exception () throw();
  exception (const exception&) throw();
  exception& operator= (const exception&) throw();
  virtual ~exception() throw();
  virtual const char* what() const throw();
}
```

exception

- Możemy tworzyć własne!

```
class myexception: public exception
{
public:
 myexception(const string& s): message(s) {}
 virtual ~myexception() throw(){}
private:
 string message;
  virtual const char* what() const throw()
  {
    return message.c_str();
  }
};
```

Proces vs Wątek

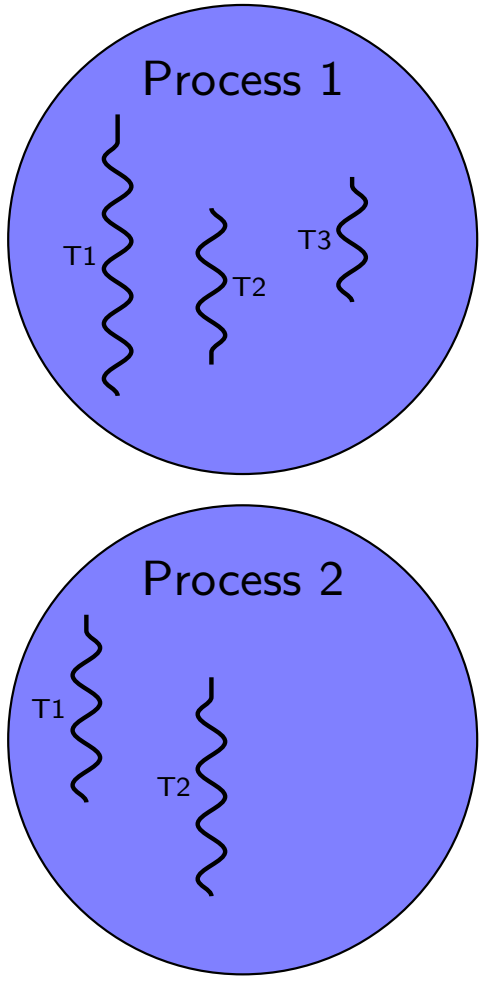
Gdy jest nas wielu

Process

- Niezależny
- Własna, niedzielona przestrzeń adresowa
- Komunikacja tylko przez system (shared memory, semaphores)

Wątki

- Istnieje w procesie
- Wspólna przestrzeń adresowa

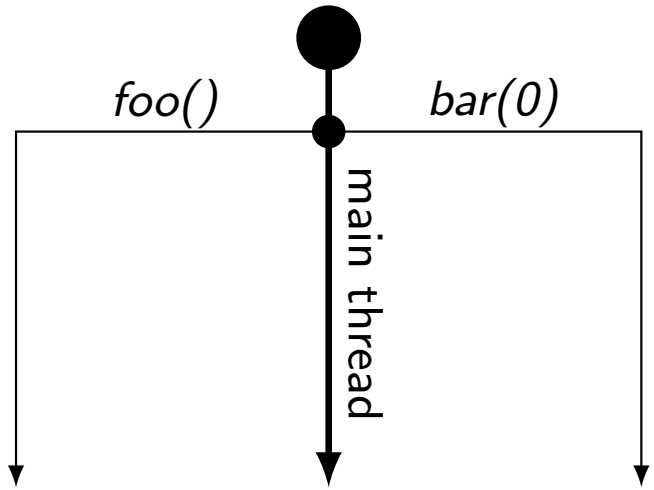


Wątek

Nowy wątek

```
#include <thread>

void foo(){
  for(int i=0; i<10; ++i){
   cout << "foo says: foo sleeps "
        << i << endl;
   sleep ( 1 );
  }
}
void bar(int x){
  for(int i=0; i<10; ++i){
   cout << "bar says: x=" << x <<
       " bar sleeps " << i <<
       endl;
   sleep ( 1 );
  }
}
...
std::thread first (foo);
std::thread second (bar,0);
```

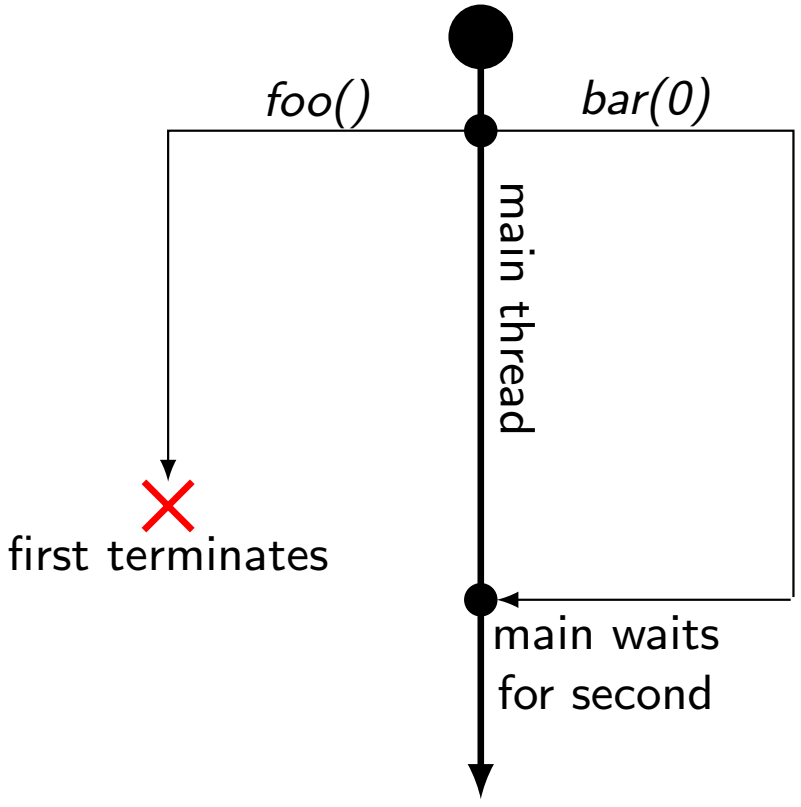


Wątki

join detach

```
#include <thread>

...
std::thread first (foo);
std::thread second (bar,0);
// detach first from main
first.detach()
...
//main thread does sth
...
// wait for second to terminate
second.join()
```



Wątki

Dostęp do zasobów

- Wątki współdzielą pamięć (i inne)
- Należy unikać 'wyścigu'
- *mutex*
- *lock* i *unlock*

Procesy
fork

- fork
- morph