Vrije Universiteit Brussel

OSSEC Project presentation:

Synchronization and Communication

Mathijs Saey,
mathsaey@vub.ac.be

Problem Domain

- Multiple processes/threads running concurrently
- Allow processes and thread to work in a *shared environment*
- Shared resources
- Need for Communication

Synchronization

- Locks & Mutexes
- Semaphores
- Condition Variables & Monitors

Communication

- Files
- Signals
- Sockets
- Message Queues
- Pipes
- Shared Memory

Examples

- Access a shared file
- Limit access to shared resource
- **Notification when a value appears**
- Find other processes
- Kill process
- Local ping server
- **Send functions to worker process**
- **Pipes & Filters**
- Data sharing

Notification on Condition

```
def consume():
    with condVar:
        while resource.count(22) is 0:
            condVar.wait()
        print("Lucky number 22 found!", resource)
```

```
def produce():
    global resource
    for ctr in range(0,100):
        with condVar:
            i = random.randint(0, 100)
            if i is 22: condVar.notify()
            resource += [i]
```

Worker Process

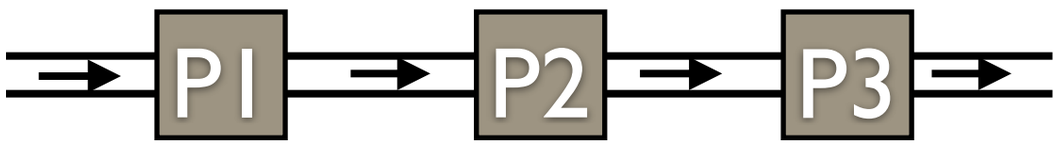
```
def sendFunction(function, args, inQueue, outQueue):
    inQueue.put((function, args)),
    return outQueue.get()
```

```
def workProcess(inQueue, outQueue):
    while True:
        tuple = inQueue.get()
        func = tuple[0]
        args = tuple[1]
        res = func(*args)
        outQueue.put(res)
```

Vrije Universiteit Brussel

Pipes & Filters

```
def createFilter(func, inc, out):
    while True:
        arg = inc.recv()
        res = func(arg)
        out.send(res)
```




```
def createProcesses(functions):
    pIn, pOut = Pipe()
    pipeLineIn = pIn

    for f in functions:
        (toNext, pEnd) = Pipe()
        p = Process(
                target = createFilter,
                args = (f, pOut, toNext))
        p.daemon = True
        p.start()
        pOut = pEnd
    return(pipeLineIn, pEnd)
```

Conclusion

- Discussed various mechanisms for both synchronisation and communication
- Example programs demonstrate the *how* and *when*

Questions?

Examples

- Access a shared file
- Limit access to shared resource
- Notification when a value appears
- Find other processes
- Kill process
- Local ping server
- Send functions to worker process
- Pipes & Filters
- Data sharing

Access Shared File

```
def writeMessage(idx, message, lock):
    message = str(idx) + " mu: " + message + '\n'
    lock.acquire()
    f = open('mutexfile.txt', 'a')
    f.write(message)
    lock.release()
```

Limited Access

```
def access(idx, sem):
    with sem:
        print(idx, "starting critical section")
        sleep(random.randint(0,3))
        print(idx, "finished critical section")
```

```
0 starting critical section
0 finished critical section
1 starting critical section
1 finished critical section
2 starting critical section
3 starting critical section
4 starting critical section
2 finished critical section
5 starting critical section
```

Find Processes

```
def discover(name, members, lock):
    leaveName(name, lock)
    print(name, "left name")
    names = []
    found = 0
    while found is not members:
        names = getNames(lock)
        found = len(names)
    print(name, "found:", names)
```

```
def leaveName(name, lock):
    name = str(name) + '\n'
    with lock:
        f = open(path, 'a')
        f.write(name)
```

```
def getNames(lock):
    with lock:
        f = open(path, 'r')
        str = f.read()
    return str.strip().split('\n')
```

Kill Process

```
def guardProcess(process, timeout):
    sleep(timeout)
    if process.is_alive():
        pid = process.pid
        os.kill(pid, getKillSig())
        print("Process killed...")
        os._exit(1)
    else:
        print("Process finished...")
        os._exit(0)
```

```
def getKillSig():
    try:
        return signal.SIGKILL
    except AttributeError:
        return signal.CTRL_C_EVENT
```

Local Ping Server

```
class requestHandler(socketserver.BaseRequestHandler):
   def handle(self):
      message = self.request.recv(BUFFER_SIZE)
      message = message.decode("utf-8")
      address = self.client_address[0]
      port = self.client_address[1]
      print("Server received:", message, "from", address, ":", port)
      self.request.sendall(bytes("pong","utf-8"))
```

```
def runClient():
   sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   sock.connect((HOST, PORT))
   sock.sendall(bytes("ping","utf-8"))
   reply = sock.recv(BUFFER_SIZE)
   reply = reply.decode("utf-8")
   print("Client received:", reply)
```

Data Sharing

```
arr = Array('i', range(5))
remainActive = Value('i', 1)

process = Process(target = runProcess, args = (arr, remainActive))
monitor = Process(target = runMonitor, args = (arr, remainActive))
```