Сопрограммы и кооперативная многозадачность

Был этот мир глубокой тьмой окутан.
Да будет свет! И вот явился Ньютон.

(Эпиграмма XVIII века)

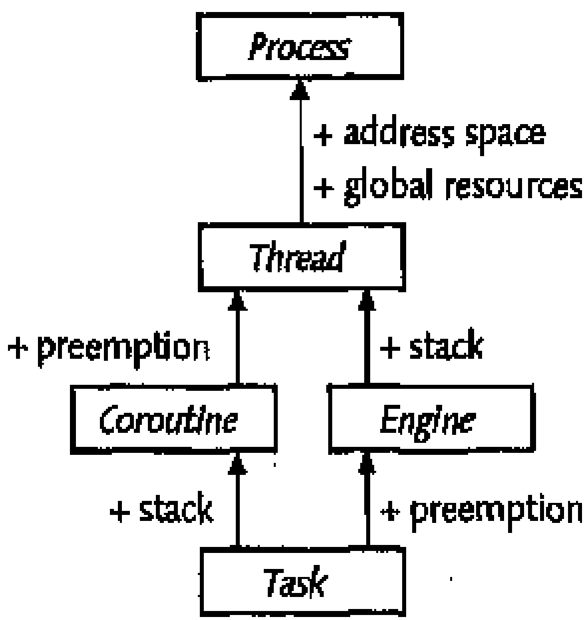
Но сатана недолго ждал реванша.
Пришел Эйнштейн - и стало все, как раньше.

(Эпиграмма XX века)

Перевод С.Маршак

Сопрограммы, потоки, процессы

Схема из C.Szyperski «Insight-EthOS»



1) Есть множество случаев, когда требуется управления задачами, по сути мета-программирование алгоритмов;
2) Есть набор технологий-артефактов, предлагающих механизмы управления задачами;
3) Деление:
    1) Task – легковесная форма;
    2) Coroutine – сопрограмма с индивидуальным стеком;
    3) Engine – процедура, вызываемая планировщиком;
    4) Thread – поток со своим стеком, вызываемый планировщиком;
    5) Process – процесс со своим адресном пространством и изоляцией данных.

Оценка угроз, а не преимуществ

- Осторожно, радиация. Ассоциация с угрозой радиоактивности, а не преимуществом ядерной технологии.

Dangling Pointers

- Отсутствие обеспечение целостности при работе с указателями.

Memory Protection

- Отсутствует изоляция данных от других задач.

Stack overflow

- Возможность переполнения стека.

LIFO calling

- Обеспечение LIFO-последовательности вызова сопрограмм.

К чему приводит?

D0 - Axiom of assignment

P0 {x := f} P

x – identifier of a simple variable;

f – an expression;

P0 is obtained from P by substituting f for all occurrences of x.

C.A.R.Hoare, An axiomatic Basis for Computer programming.

If we wish to prove correctness in more realistic world … have to satisfy for the proper execution of the program, i.e. the **properties on which the correctness proof relies** (например, exact integer arithmetic in the range, или правильная работа с указателями при арифметических вычислениях).

Dijkstra, Notes on Structural Programming (6. On the validity of proofs vs the validity of implementations).

Memory Protection

А если Нет атомарности присваивания?

Dangling Pointers

А если Не обеспечена целостность указателей?

Проблемы как следствия

- Не выполняются необходимые условия, на которых основывается доказательство корректности, используемой для построения структурной программы по Дейкстре;
- Неверны аксиомы, используемые для доказательства частичной корректности логики Хоара;

$\Rightarrow$Эволюция или Инволюция от структурного программирования к неструктурному?

$\Rightarrow$Доказательность заменяется объемным тестированием, результат – редко (закон больших чисел) возникающие ошибки.

Co_, Coroutines

Продать «смотрите, что я ради вас сделал» куда легче, чем «смотрите, от чего я ради вас уклонился».

(Нассим Талеб, Антихрупкость)

- Co_ - кроссплатформенный пакет (Д.Дагаев, 2013);
- Реализация сопрограмм и кооперативной многозадачности (разновидности процесса внутри самого процесса).

ОС	**BB**	**XDS**	**Ofront**
Windows	Fiber	COROUTINES	
Linux	ucontext_t	COROUTINES	ucontext_t

- Coroutines (J.Templ, 2017) – пакет для BlackBox 1.7.1;
- “issue-#156 adding Coroutines to BlackBox”;
- Реализация сопрограмм.

Простой пример

```
Generator = POINTER TO RECORD
   (Co.CoroutineDesc) v: INTEGER END;
PROCEDURE (g: Generator) Do;
BEGIN
   g.v := 1; Co.Yield;
   g.v := 2; Co.Yield;
   g.v := 3; Co.Stop;
END Do;

PROCEDURE Run*;
   VAR g: Generator;
BEGIN
   NEW(g);
   g.Start;
   WHILE ~g.eor DO
        g.Transfer;
        Log.String("Gen"); Log.Int(g.v);
   Log.Ln
   END;
   Log.String("End");Log.Ln;
END Run;
```

На выходе:

Gen 1

Gen 2

Gen 3

End

Гарантии указателей

Dangling Pointers

BB|Windows:

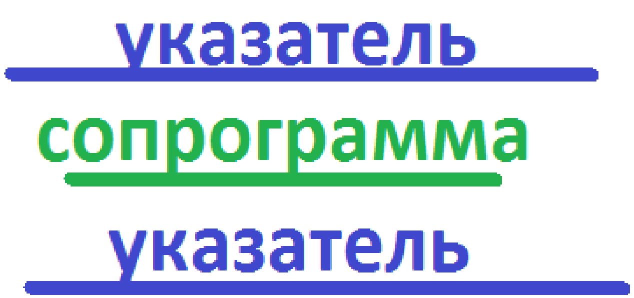
J.Templ модификация runtime BlackBox 1.7.1, алгоритм сборки мусора с учетом локальных стеков

Dangling Pointers

BB|Linux:

Требует модификации BlackBox Kernel

Условие купирования – время жизни сопрограммы внутри времени жизни указателя




```
PROCEDURE (dict: Dict) Do ;
   VAR list, li: List;
BEGIN
   NEW(list); list.name := Name();
   AddNext(dict, list);
   li := list;
   Co.Yield
END Do;
```

Same Fringe Problem Дейкстры

Сравнение «бахромы» двух деревьев:

T(l(a,b), l(c,nil))

T(l(a,l(nil,b)), l(l(c,nil),nil))

Результат:

abc = abc

Нужно 2 стека с рекурсией и выдачей промежуточных результатов.

NB! Унификация ПРОЛОГ машины на схожих принципах.

```
PROCEDURE NextTreeLeaves (trav: Traverse; tree: Tree);
BEGIN
    IF (tree.left = NIL) & (tree.right = NIL) THEN
           trav.leaf := tree; Co.Yield
    END;
    IF tree.left # NIL THEN
           NextTreeLeaves (trav, tree.left)
    END;
    IF tree.right # NIL THEN
           NextTreeLeaves (trav, tree.right)
    END;
END NextTreeLeaves;
PROCEDURE (trav: Traverse) Do ;
BEGIN
    NextTreeLeaves(trav, trav.tree); Co.Stop
END Do;

NEW(r1); r1.tree := t1;
NEW(r2); r2.tree := t2;
same := TRUE;
r1.Start; r2.Start;
WHILE ~r1.eor & ~r2.eor & same DO
    r1.Transfer;
    r2.Transfer;
    same := (r1.leaf.name$ = r2.leaf.name$)
END;
IF ~r1.eor OR ~r2.eor THEN same := FALSE END;
```

Защита стеков сопрограмм

```
PROCEDURE Test (limit: INTEGER);
    VAR buffer: ARRAY 1024 OF CHAR;
BEGIN
    IF limit > 0 THEN
           Test(limit-1)
    END
END Test;
PROCEDURE (st: StackTest) Do ;
BEGIN
    Test(st.limit)
END Do;
PROCEDURE Run*;
    VAR st: StackTest;
BEGIN
    NEW(st);
    st.limit := 1000000;
    st.Start;
    st.Transfer;
END Run;
```

Stack overflow

BB Windows Fiber

Stack overflow

BB Linux ucontext_t

Условие купирования – отсутствие или ограничение рекурсии

Инвертированное программирование

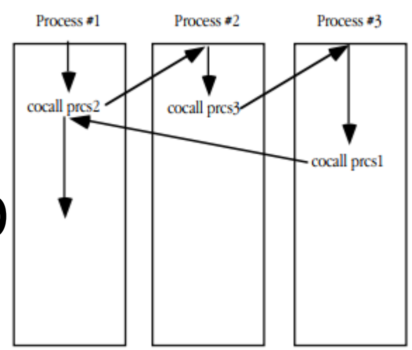
Из книги Дональда Кинга "Создание Эффективного Программного Обеспечения". Кинг использует методику М.Джексона из "Principles of Program Design".

В примере ReadBuffer и ReadString выступают как производитель/потребитель. ReadBuffer заполняет буфер размером BUFSIZE и уступает управление ReadString. Когда ReadString снова опустошает буфер, ReadBuffer возобновляет свою работу с состояния на момент останова. В состояние входят имя текущего файла, его хэндлер и позиция в файле, все это хранится в стеке локальных переменных.

- "Co_ObxMJackson.Grep('start', 'StartupBlackBox.vbs \Windows\setupact.log')"
- "Co_ObxMJackson.PipeInit('StartupBlackBox.vbs \Windows\setupact.log')"
- Co_ObxMJackson.More

2 типа сопрограмм

- Directed – вызываются из другой сопрограммы/главной программы с передачей и/или приемом параметров и обеспечением процедуры LIFO
- Anonymous – вызываются без передачи/приема параметров планировщиком кооперативной многозадачности

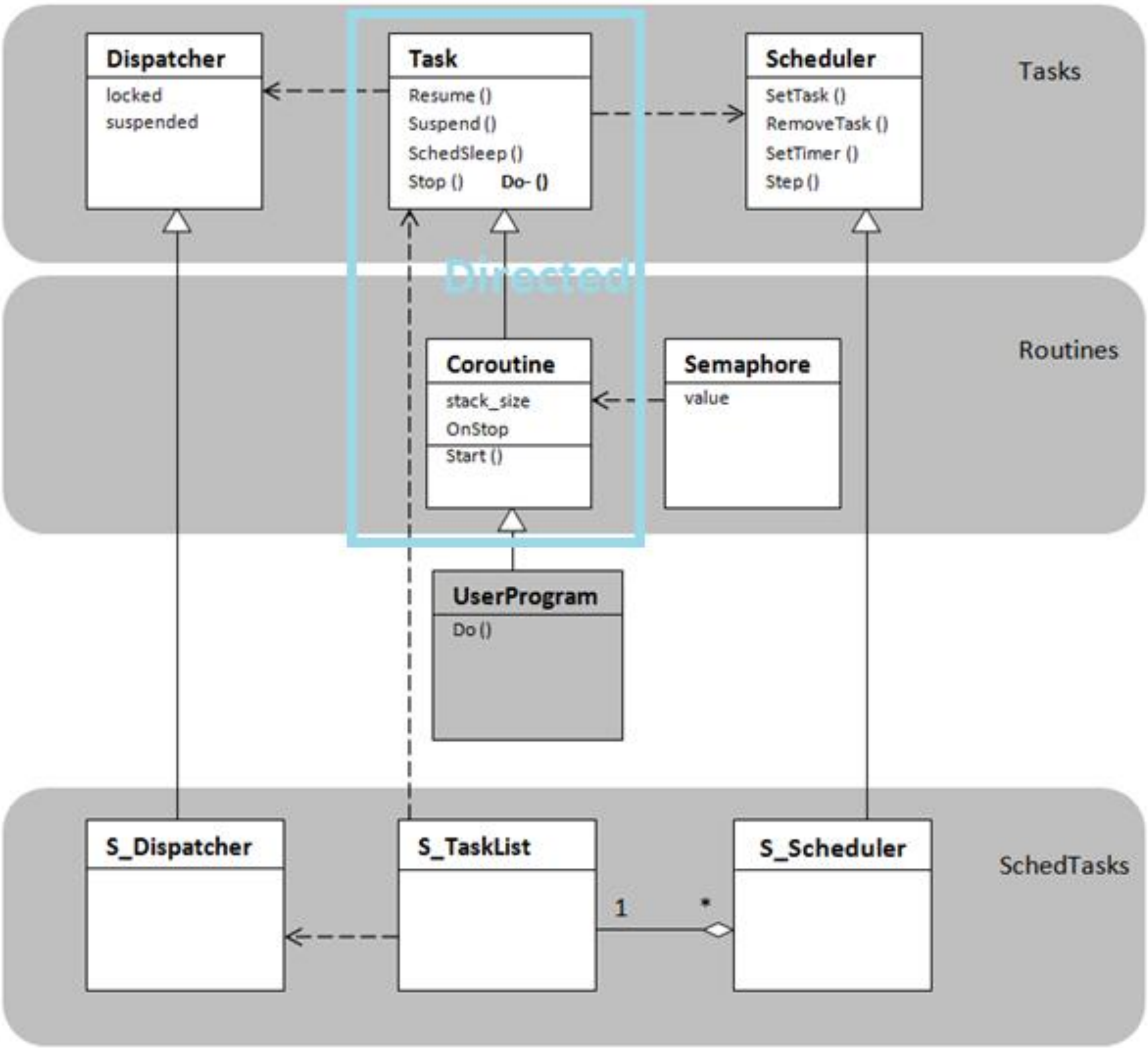


Cocalls Between Three Processes

Co_ обеспечено на уровне рантайма ASSERTами

Coroutines J.Templ не обеспечено, попытка сделать возможным Anonymous-вызовы для будущего

Co_ с многозадачностью сложнее, чем Coroutines без



- Объектная модель и планировщик.
- Directed сопрограммы используют лишь 2 типа 2 модулей.

Задача в фоновом режиме

```
PROCEDURE (a: PrimeAction) Do ;
  VAR end, sqrtCur, n: INTEGER;
BEGIN
  FOR n := 0 TO a.attempts-1 DO
    a.current := 3; a.divisor := 3;
    REPEAT
      end := a.divisor + stepSize;
       sqrtCur := SHORT(ENTIER(Math.Sqrt(a.current)));
       WHILE (a.divisor <= sqrtCur) & (a.divisor < end)
       & (a.current MOD a.divisor # 0) DO
         IF a.divisor MOD 256 = 1 THEN Co.Yield END;
         a.divisor := a.divisor + 2
       END;
       IF a.divisor > sqrtCur THEN (* cur is a prime *)
         a.divisor := 3; a.current := a.current + 2
       ELSIF a.divisor < end THEN
         (* not a prime, test next *)
         a.divisor := 3; a.current := a.current + 2
       END
     UNTIL a.current > a.to;
  END;
  Co.Stop
END Do;

s := Ct.scheduler;
p.interval := 50;
p.load_pct := 100;
s.SetParams(p);
```

Планировщик вызывает процедуру Do в фоновом режиме, когда есть свободное процессорное время.

1. Пример BlackBox ObxActions вызывает процедуру Do 1 раз за цикл Services.Action;
2. В примере Co_ObxActions планировщик вызывает процедуру Do столько раз, сколько можно за период (у меня – 400000).

Производитель-потребитель

+Co_ -Coroutines

Сообщения планировщику передаются через семафоры;

Memory Protection

Сопрограммы переключаются в безопасных точках разрыва Yield

Memory Protection

Многопоточные приложения не имеют универсального механизма защиты.

Каждый блок данных защищаем явно.

```
PROCEDURE Send (VAR mb: MessageBox; VAR msg: Ct.Message);
BEGIN
	Co.SemDown(mb.sempty);
	WITH msg: Message DO
		mb.msg := msg
	| msg: AckMessage DO
		mb.ack := msg
	END;
	Co.SemUp(mb.sdata)
END Send;

PROCEDURE Recv (VAR mb: MessageBox; VAR msg: Ct.Message);
BEGIN
	Co.SemDown(mb.sdata);
	WITH msg: Message DO
		msg := mb.msg
	| msg: AckMessage DO
		msg := mb.ack
	END;
	Co.SemUp(mb.sempty)
END Recv;
```

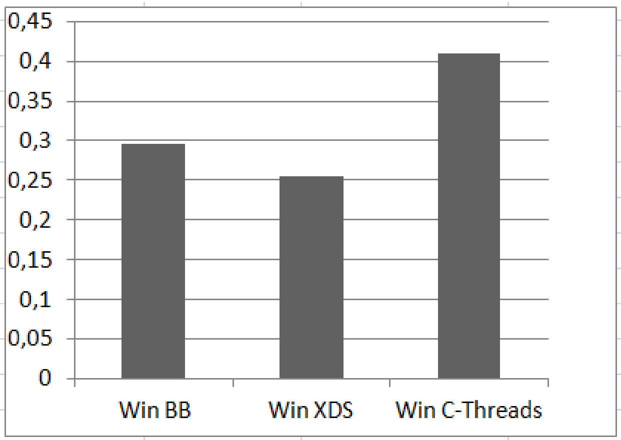
Обедающие философы Дейкстры

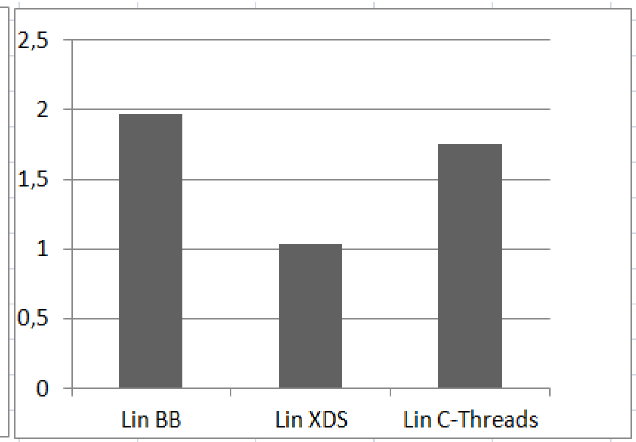
+Co_ -Coroutines

Знаменитая задача Дейкстры на разделение доступа. Вилки являются ограниченным ресурсом, философ может приступить к еде, только имея 2 вилки. Вилки защищены семафорами.

THINKING.THINKING.THINKING.THINKING.HUNGRY
THINKING.THINKING.THINKING.THINKING.EATING
THINKING.THINKING.THINKING.HUNGRY.EATING
THINKING.THINKING.HUNGRY.HUNGRY.EATING
THINKING.THINKING.EATING.HUNGRY.EATING
THINKING.HUNGRY.EATING.HUNGRY.EATING
HUNGRY.HUNGRY.EATING.HUNGRY.EATING
HUNGRY.HUNGRY.EATING.HUNGRY.THINKING
EATING.HUNGRY.EATING.HUNGRY.THINKING
EATING.HUNGRY.THINKING.HUNGRY.THINKING
EATING.HUNGRY.THINKING.EATING.THINKING
THINKING.HUNGRY.THINKING.EATING.THINKING
THINKING.EATING.THINKING.EATING.THINKING

Сравнительное время переключения





Среднее время переключения между задачами, мкс

Были оценены времена переключений между сопрограммами по сравнению с временами переключения между потоками в реализации на C для Windows/Linux в тестовом примере с числом задач = 100.

Об использовании заимствованного ПО

Во время работы у Эдисона я обратил внимание на его стремление доводить до конца любые проекты и делать это как можно быстрее. Даже если в ходе работы над созданием какого-либо устройства выяснялось, что в таком виде оно никуда не годится и лучше пойти другим путем (с большими затратами времени), Эдисон требовал дать ему результат как можно скорее. То есть, вместо того, чтобы сделать работу один раз как следует, её делали как придётся, лишь бы скорее, а потом переделывали несколько раз. При этом тратилось больше времени, усилий и денег. Казалось бы, Эдисон поступал нелогично. Я так и думал до тех пор, пока мне не объяснили, в чем дело. 90% работ Эдисона финансировалось со стороны. Заказчики большей частью ничего не понимали в электротехнике. Они знали, что это выгодное вложение и больше ничего знать не желали. Они не могли понять тонкостей, им нужен был только результат. Чем скорее будет результат, тем прочнее деловая репутация Эдисона и всей его компании.

Никола Тесла, из дневников.