Л.Ф. Гореликов

Опыт разработки и внедрения в эксплуатацию системного программного обеспечения управляющей БЦВМ КА

Module RESEARCH CENTRE

Октябрь 2021 г.

О компании НТЦ Модуль

Основана

1990

Число сотрудников

650 +

Проектирование интегральных микросхем (услуги микроэлектронного дизайна)

Проектирование и производство специальных вычислительных модулей и блоков (бортовая космическая и авиационная аппаратура)

Производство и проектирование систем распознавания и анализа видеоизображений

Программно-аппаратные решения для задач AI на базе собственной архитектуры нейропроцессора NeuroMatrix®

Module
RESEARCH CENTRE

Зачем мы делали системное ПО для БЦВМ КА?

РОСКОСМОС

НПО ИМЕНИ
С.А.ЛАВОЧКИНА

Module
RESEARCH CENTRE

Что входит в понятие «системное ПО» для БЦВМ КА?

- BIOS
- Bootloader
- RTOS + BSP
- IDE
- Эмулятор БЦВМ

ПО для БЦВМ КА относится к классу Safety-Critical Real-Time Software

Module RESEARCH CENTRE

2013 - 2021…

Module
RESEARCH CENTRE

RTOS: Статистика

- ~1 млн. строк исходного кода
- Код критически важных модулей BIOS, bootloader, RTOS и BSP покрыт тестами на 100%, общее покрытие порядка 85% (покрытие по метрике Statement Coverage)
- Пройдены лабораторно-отработочные испытания, предварительные испытания, автономные испытания, комплексные испытания, проведено квалификационное тестирование

Module RESEARCH CENTRE

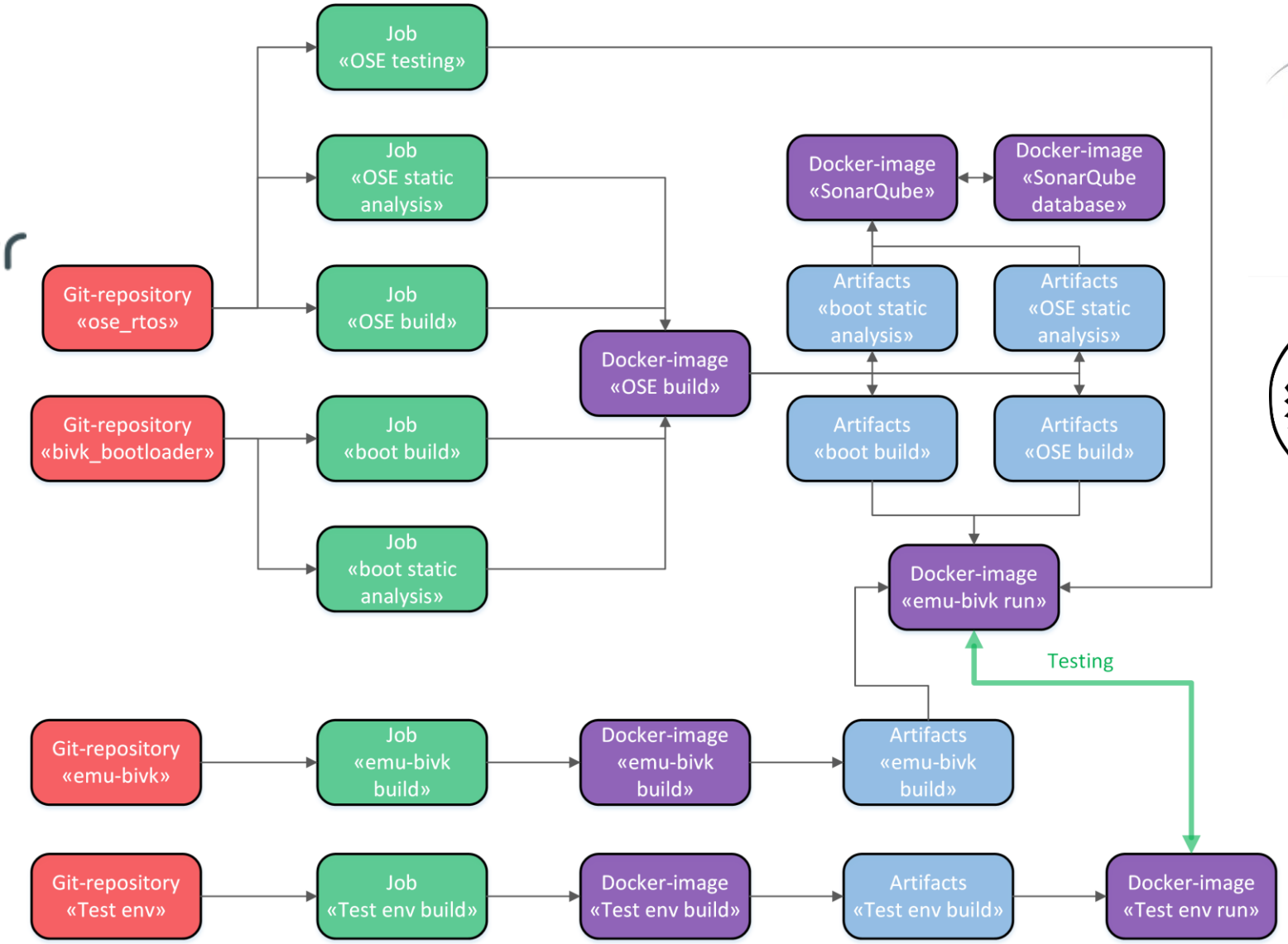
RTOS: Статистика

- ~3k тестов, длительность прогона ~30 часов
- Осуществляется подготовка лётной версии СПО
- Команда ~5 человек
- 6 лет активной разработки
- +Параллельные проекты

Как удалось выполнить такой объём работы?

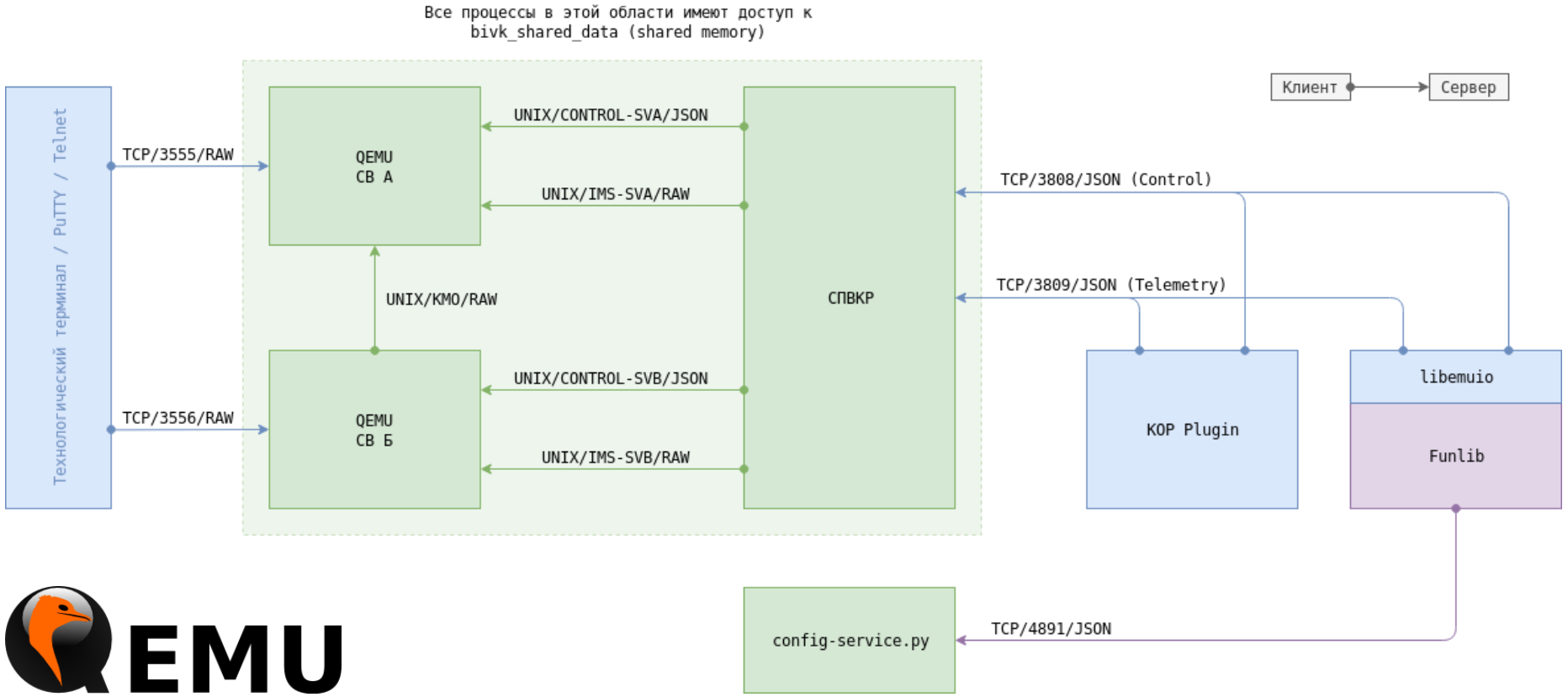
Module RESEARCH CENTRE

CI-инфраструктура



Module RESEARCH CENTRE

Полносистемный QEMU-based эмулятор БЦВМ (архитектура)



Подробнее по теме эмулятора в прошлом докладе:

https://0x1.tv/20191206AE

Полносистемный QEMU-based эмулятор БЦВМ (интеграция)

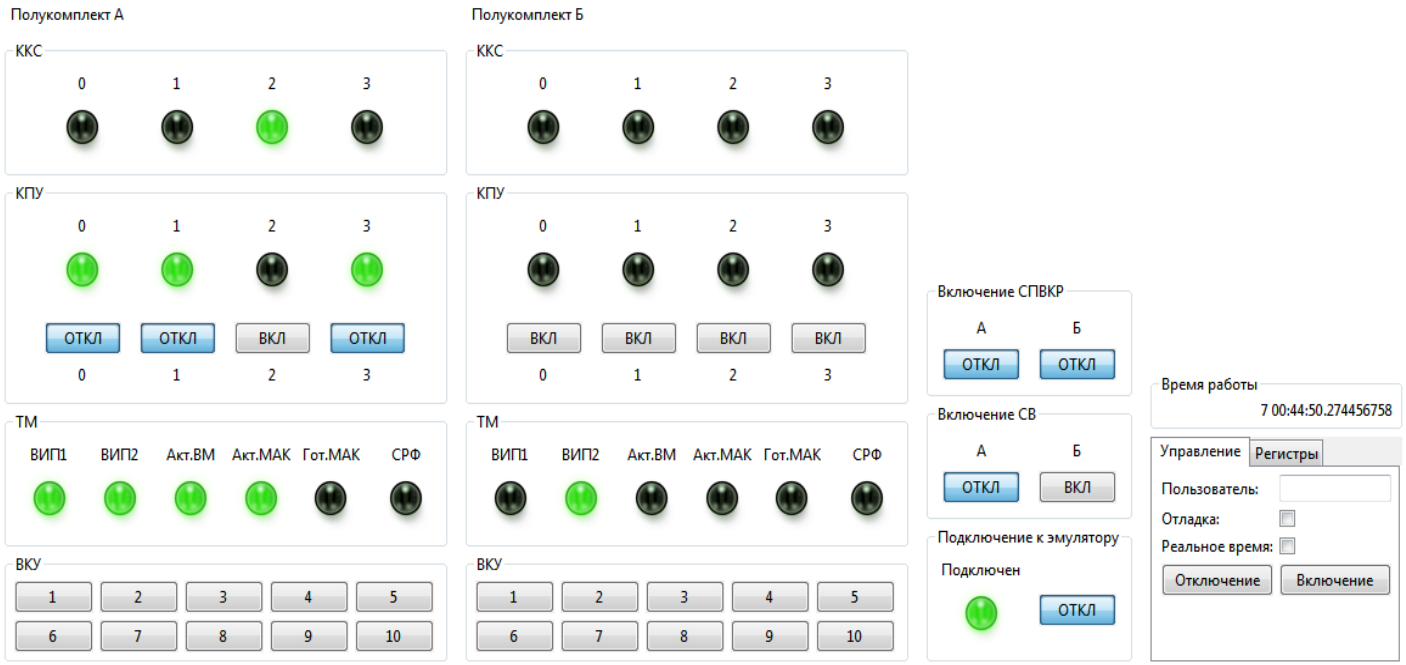
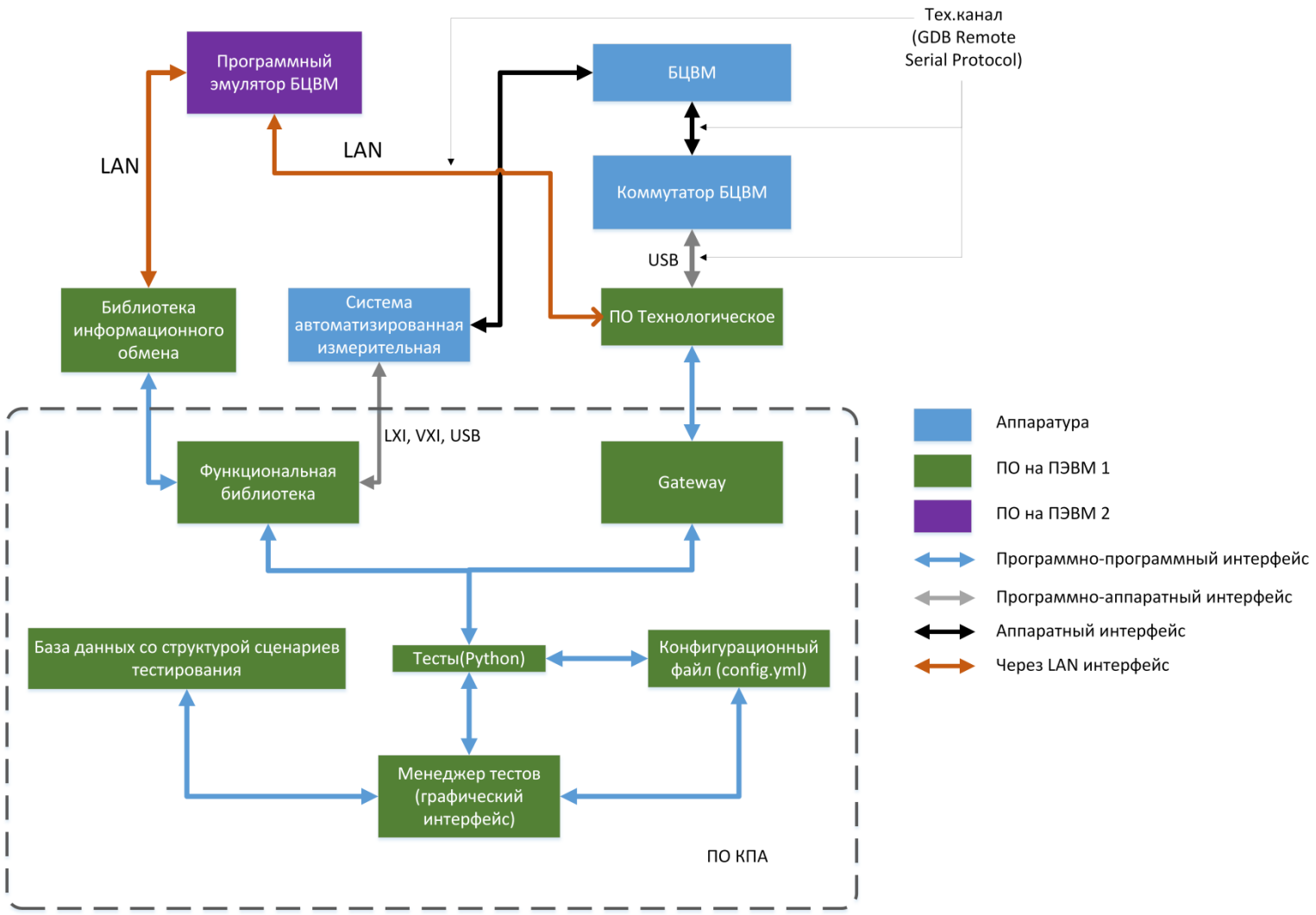
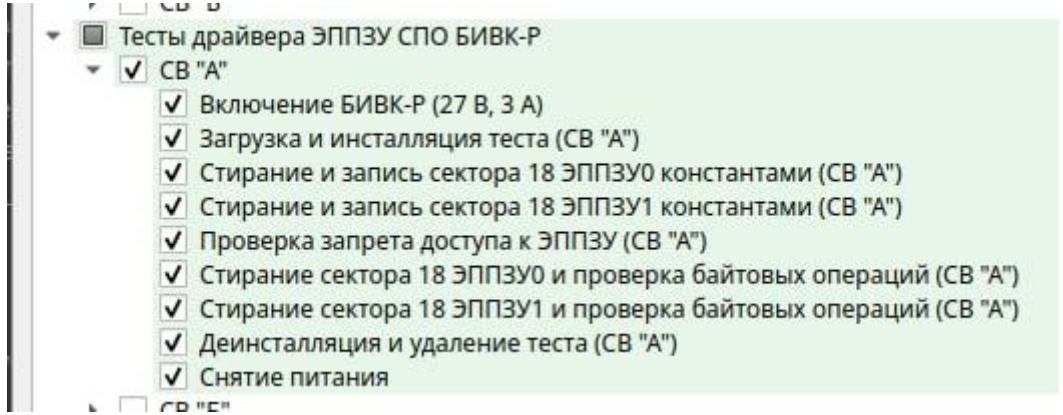


Схема взаимодействия тестового окружения и объекта контроля



Автоматизация тестового окружения (менеджер тестов)



Нестандартные ошибки

Module
RESEARCH CENTRE

RTOS: Модульность - главное преимущество микроядер для safety critical software

- Модульность позволяет гибко компоновать необходимую конфигурацию ядра (т.е. ничего лишнего)
- Отсутствие overhead в `.text` снижает трудоёмкость верификации (более структурированные тесты, легче достигнуть требуемых метрик покрытия и т.д.)
- Возможность разнести разные функциональные сущности в отдельные домены памяти, что позволяет повысить надежность системы в случае повреждения памяти

Module RESEARCH CENTRE

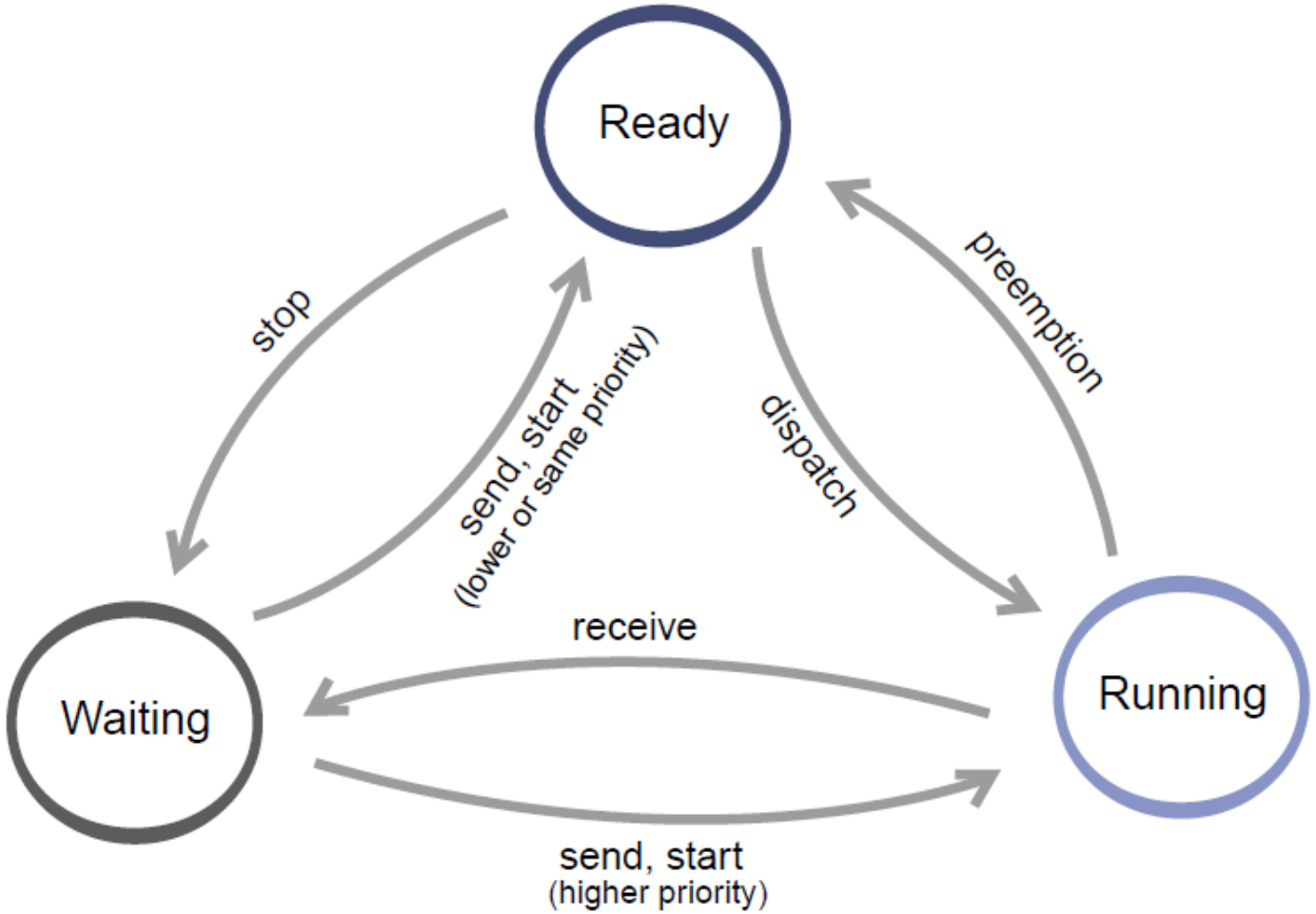
RTOS: Инверсия приоритетов - главный недостаток микроядер для real-time software

В случае применения алгоритмов планирования задач с статическими приоритетами (rate-monotonic scheduling (RMS), deadline-monotonic…) инверсия приоритетов приведёт к недетерминированному отклику системы (возможен «отказ в обслуживании»)

В случае применения алгоритмов планирования задач с динамическими приоритетами (earliest deadline first (EDF), least slack time (LST)…) инверсия приоритетов приведёт к увеличению фазового дрожания (jitter) процессов, которое необходимо оценить

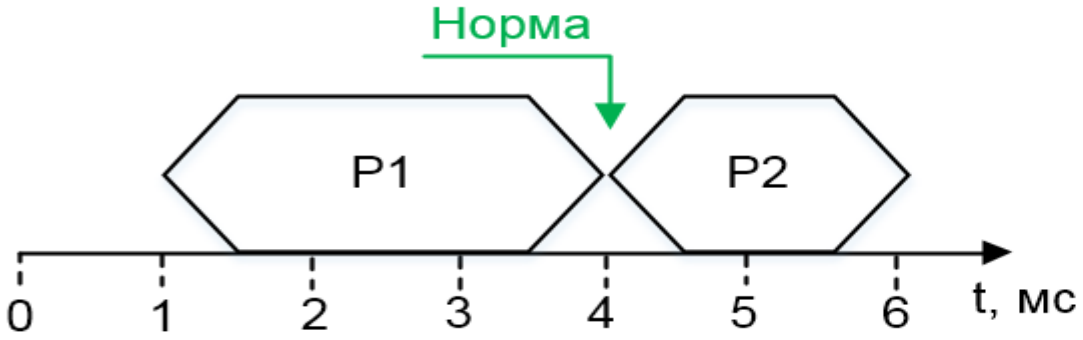
Module RESEARCH CENTRE

RTOS: Механизм возникновения инверсии приоритетов в микроядре

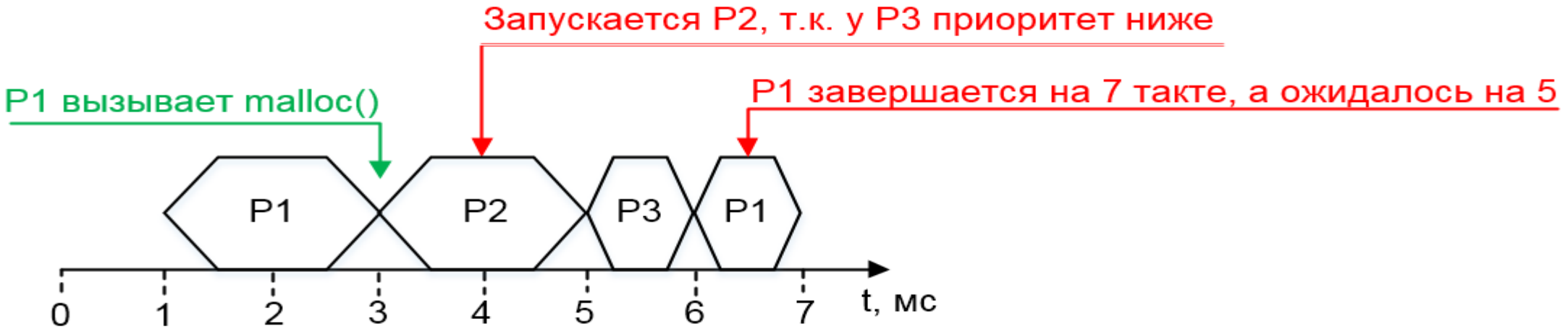


RTOS: Механизм возникновения инверсии приоритетов в микроядре

P1: приоритет - 0(макс.), длительность – 3 кванта
P2: приоритет - 5, длительность – 2 кванта
P3: приоритет - 10(мин.), длительность – 1 квант



1 квант = 1 мс
P3 – Memory manager



RTOS: Инверсия приоритетов - главный недостаток микроядер для real-time software

Полностью гарантировать отсутствие инверсии приоритетов в данном случае возможно лишь путём формальной верификации взаимодействия всех процессов в системе методами model checking

Для формальной верификации взаимодействия процессов системных служб, драйверов, отложенных вложенных обработчиков прерываний мы используем инструменты SPIN и UPPAAL

Module
RESEARCH CENTRE

RTOS: Bug при работе с FPU-конвейером

```
void test_fon(void)
{
 int i=0;
 union { unsigned int w[2]; double d; } u;
 static double fP=0.0;
 unsigned long res = 0;
 Msr msr;

while(1)
 {

 if(fP!=0.0)
 {
 u.d=fP;
 printf("--------- %08x%08x\n", u.w[1],u.w[0]);

 }


if(((++i)&0xffff)==0) printf("i=%d\n",i>>16);
 }

}



void test_proc(void)
{
 int i; double fP,fM,fD,sq,sn,cs;
 fP=0.; fM=1.; fD=1.;

 for(i=0;i<100;i++)
 {
 fP+=1./3.; fM*=1.001; fD/=1.0001; sq=sqrt(fP); sn=sin(fP*10.); cs=cos(fP*10.);
 }
}
```

0 != 0 ???

Module RESEARCH CENTRE

RTOS: Bug при работе с FPU-конвейером

```
2135
2136  #       Swap out fp registers.
2137          cfc1    t5,$31
2138
2139          mfc0    t2,C0_SR
2140          and     t2,~SR_IE
2141          mtc0    t2,C0_SR
2142
2143          cfc1    t2,$25
2144          cfc1    t3,$26
2145          cfc1    t4,$28
2146          cfc1    t5,$31
2147
2148          /*
2149          and t5,~0x7C
2150          ctc1    zero,$31
2151          */
2152
2153          sw      t2,fpu_fcr25(t1)
2154          sw      t3,fpu_fcr26(t1)
2155          sw      t4,fpu_fcr28(t1)
2156          sw      t5,fpu_fcr31(t1)
```

MIPS ISA: FCR31 must only be written to when the FPU is not actively executing floating-point operations; this can be ensured by reading the contents of the register to empty the pipeline.

Module RESEARCH CENTRE

RTOS: Error handler bug

Обработка системного вызова error() режиме ядра:

1. error() (без syscall, т.к. это контекст ядра)
2. вызов error_handler процесса, который возвращает 0
3. ядро проверяет код возврата, понимает что он 0 и вызывает error_handler блока (т.е. следующий уровень), который возвращает 1
4. обработка error() завершена - Норма

RTOS: Error handler bug

Обработка системного вызова error() режиме пользователя:

1. error() (уже с syscall)
2. ядро обрабатывает системный вызов (через исключение процессора) и вызывает error_handler процесса (уже в контексте пользователя, т.е. после инструкции eret), который возвращает 0
3. код возврата обрабатывает ядро, поэтому снова вызывается syscall
4. в процессе переключения в контекст ядра код возврата пользовательского error_handler затирается - Ошибка

RTOS: Error handler bug

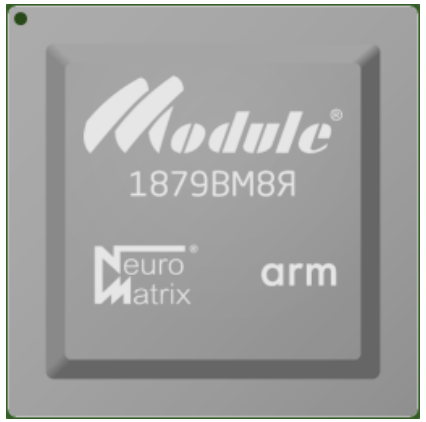
```
lw t8,branch_pointer
move t6,sp
lw v0,handle
j t8 #Branch to kernel or to syscall.
move t7,ra
```

Какие из этого можно сделать выводы?

Module
RESEARCH CENTRE

Планы

- 2022 – долететь до Луны (см. КА Луна-Глоб)
- 2023 – построить гетерогенную масштабируемую БЦВМ с контуром нейрообработки (64 ядра NMC4, 20 ядер ARM A5, 1 ядро PowerPC 476 FP, 20 Гб DDR3, 2 Тб SSD) и поддержкой OpenCL, OpenCV, OpenGL.
- 2024 – ещё раз долететь до Луны (см. КА Луна-Ресурс) и закончить БЦВМ для КА ДЗЗ



Module RESEARCH CENTRE

Спасибо за внимание

l.gorelikov@module.ru

lev-gorelikov@mail.ru

https://www.researchgate.net/profile/Lev-Gorelikov

RESEARCH CENTRE

www.module.ru